DOCUMENT RESUME

ED 383 548 SE 056 253

AUTHOR Thomas, David A., Ed.
TITLE Scientific Visualization in Mathematics and Science Teaching.
INSTITUTION Association for the Advancement of Computing in Education, Charlottesville, VA.
REPORT NO ISBN-1-880094-09-6
PUB DATE 95
NOTE 293p.
AVAILABLE FROM AACE, P.O. Box 2966, Charlottesville, VA 22902.
PUB TYPE Books (010) -- Guides - Classroom Use - Teaching Guides (For Teacher) (052) -- Collected Works - General (020)

EDRS PRICE MF01/PC12 Plus Postage.
DESCRIPTORS Calculators; Computer Software; *Computer Uses in Education; *Educational Technology; Elementary Secondary Education; Graphs; *Mathematics Instruction; *Science Instruction; *Technology Education; *Visualization
IDENTIFIERS Graphing Utilities

ABSTRACT

Science and mathematics educators are expected to use existing educational technologies effectively and to keep informed about emerging technologies that might become important educational tools in the not-so-distant future. This monograph offers some help in that regard by highlighting a number of existing and emerging educational technologies. Chapters are: (1) "The Power of Visualization: The Impact of Graphing Technology on the Secondary Mathematics Curriculum," L. E. Yunker; (2) "Using Graphing Calculators to Teach High School Mathematics," L. Kaber & K. Longhart; (3) "Advanced Technologies as Educational Tools in Science: Concepts, Applications, and Issues," D. D. Kumar, P. J. Smith, S. L. Helgeson, & A. L. White; (4) "Videodisc Technology: Applications for Science Teaching," D. R. Lavoie; (5) "Computer Visualization: New Window on Mathematics," D. A. Thomas & M. Mitchell; (6) "Visualizing Computer Science," R. J. Ross; (7) "Getting Started With Supercomputing: An Approach for High School Students," D. W. Hyatt; (8) "Scientific Visualization in Chemistry, Better Living through Chemistry, Better Chemistry through Pictures: Scientific Visualization for Secondary Chemistry Students," R. R. Gotwals, Jr.; (9) "The National Education Supercomputer Program," R. Enderton & B. Lindow; (10) "New Mexico High School Supercomputing Challenge," M. S. Foster; (11) "Sharing Multiple Complementary Representations in the Teaching of Science," N. H. Sabelli & I. S. Livshits; (12) "Education and Collaboration in an Evolving Digital Culture," D. J. Cox; (13) "The Hypergraphics Honors Seminar at Illinois," G. K. Francis; and (14) "A Syllabus For Scientific Visualization," A. Pang. (MKR)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 383 548

SE

Science Teaching

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it

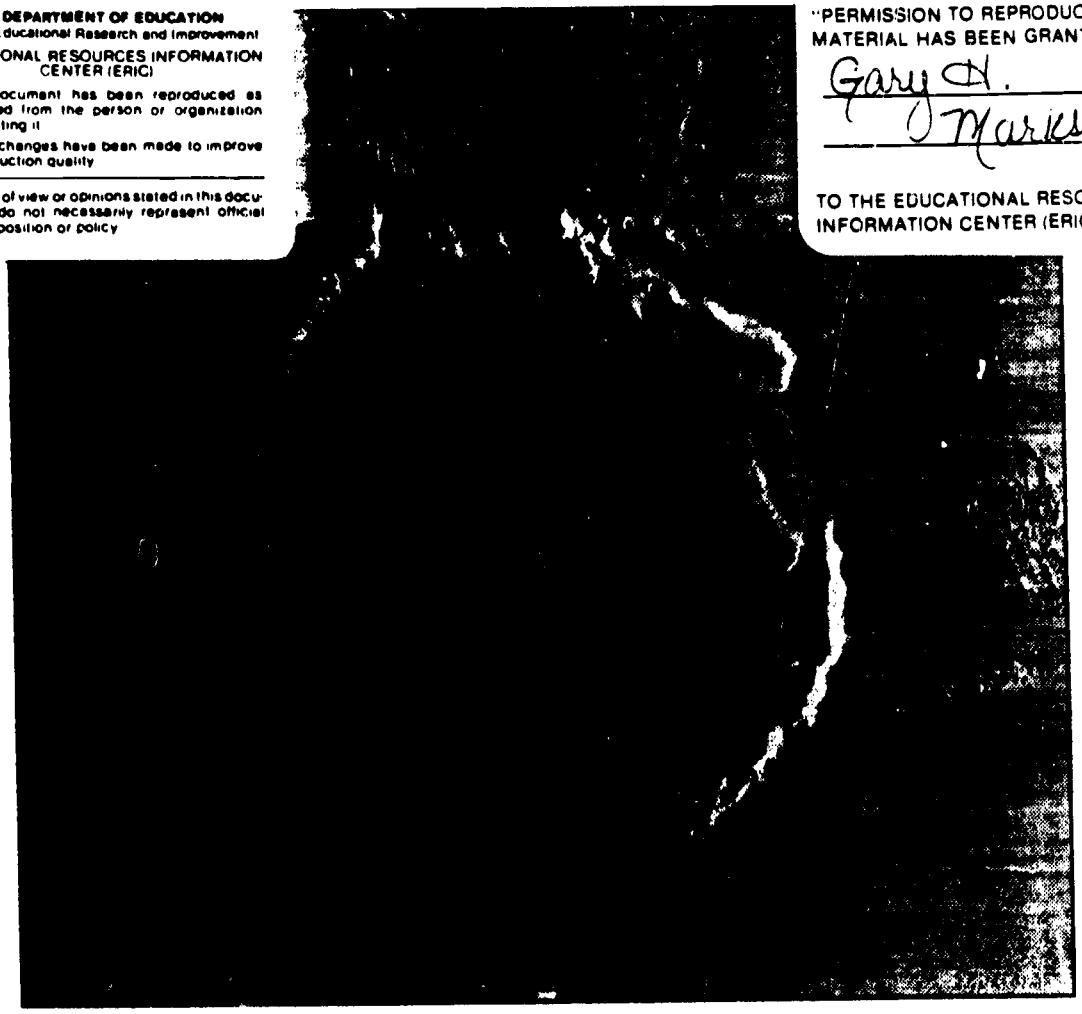
☐ Minor changes have been made to improve reproduction quality

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Gary A. Marks

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)"



edited by David A. Thomas

BEST COPY AVAILABLE

Scientific Visualization in Mathematics and Science Teaching

edited by David A. Thomas

Association for the Advancement of Computing in Education

ISBN 1-880094-09-6

Copyright © 1995 by the Association for the Advancement of Computing in Education

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without permission in writing from the publisher.

The publisher is not responsibile for the use which might be made of the information containing in this book.

Published by the Association for the Advancement of Computing in Education (AACE) P.O. Box 2966, Charlottesville, VA 22902 USA

Printed in the USA

CONTENTS

Introduction i

Contributors iii

Chapters

1 **The Power of Visualization: The Impact of Graphing Technology on the Secondary Mathematics Curriculum**
LEE E. YUNKER 1

2 **Using Graphing Calculators to Teach High School Mathematics**
LARRY KABER AND KAREN LONGHART 19

3 **Advanced Technologies as Educational Tools in Science: Concepts, Applications, and Issues**
DAVID D. KUMAR, PHILIP J. SMITH, STANLEY L. HELGESON, AND ARTHUR L. WHITE 27

4 **Videodisc Technology: Applications for Science Teaching**
DERRICK R.LAVOIE 45

5 **Computer Visualization: New Window on Mathematics**
DAVID A. THOMAS AND MARK MITCHELL 67

6 **Visualizing Computer Science**
ROCKFORD J. ROSS 99

7 **Getting Started With Supercomputing: An Approach for High School Students**
DONALD W. HYATT 129

8 **Scientific Visualization in Chemistry, Better Living Through Chemistry, Better Chemistry Through Pictures: Scientific Visualization for Secondary Chemistry Students**
ROBERT R. GOTWALS, JR. 153

9 **The National Education Supercomputer Program**
RICHARD ENDERTON AND BRIAN LINDOW 181

10 **New Mexico High School Supercomputing Challenge**
MARILYN S. FOSTER 201

11 Sharing Multiple Complementary Representations in the Teaching of Science
NORA H. SABELLI AND IGOR S. LIVSHITS 213

12 Education and Collaboration in an Evolving Digital Culture
DONNA J. COX 225

13 The Hypergraphics Honors Seminar at Illinois
GEORGE K. FRANCIS 237

14 A Syllabus For Scientific Visualization
ALEX PANG 261

Color Prints 285

Cover: Olympos Mons, Mars

Introduction

Not so long ago, educators prided themselves on the stability of their programs and the traditions of their institutions. Formal education's principal goals were to foster an appreciation for learning and to develop citizens of sound moral character and judgment. For better or for worse, today's fast-paced world is forcing educators and educational institutions to refocus their goals to accommodate or even anticipate change. For instance, as science and mathematics educators, we are expected to use existing educational technologies effectively and to keep informed about emerging technologies that might become important educational tools in the not-so-distant future. This monograph offers some help in that regard by highlighting a number of existing and emerging educational technologies. Chapters were contributed by classroom teachers, university mathematics and science educators, and specialists from the National Science Foundation (NSF), the National Center for Supercomputing Applications, and a number of other governmental agencies.

In chapters 1 and 2, the use of graphing calculators in high school mathematics is discussed. This technology offers a low-cost alternative to computer-based graphing packages. In what ways does this technology support existing goals advanced by the NCTM *Curriculum and Evaluation Standards*? How should a school approach the problems of cost and equity associated with this technology? What do parents and students think about the technology? These and other questions are addressed by three high school teachers who are also leading proponents of this technology.

Two popular educational technologies are discussed in chapters 3 and 4, hypermedia and interactive videodisk. What distinctive advantages do these interactive media offer science educators interested in tools for enhancing concept development? What are the costs associated with the use of these technologies? How do you get started? The authors of these chapters are all university-based science educators with extensive experience in the development and use of educational materials based on these technologies.

In chapters 5-14, a number of emerging technologies and their educational implications are discussed. The technologies range from computer microworlds to supercom-

puting and scientific visualization tools. Here, a broad spectrum of classroom teachers, university mathematics and science educators, and scientists explore exotic technologies, the nature of collaborative and interdisciplinary science in the information age, and opportunities for students and teachers interested in high performance computing and communications.

If you are interested in current and emerging educational technologies for science and mathematics education, this monograph will introduce you to a group of teachers and researchers who share your interest and who are developing and testing educational materials based on those technologies. On behalf of the authors of this monograph, welcome to the future!

David A. Thomas
Associate Professor of Mathematics Education
Department of Mathematical Sciences
Montana State University
Bozeman, MT 59717
umsfdtho@mathfs.math.montana.edu

Dedicated to

Lee E. Yunker in recognition of a lifetime of teaching and service.

Contributors

Chapter 1

The Power of Visualization: The Impact of Graphing Technology on the Secondary Mathematics Curriculum

Lee E. Yunker
Mathematics Department
West Chicago Community High School
West Chicago, Illinois 60185

Chapter 2

Using Graphing Calculators to Teach High School Mathematics

Karen Longhart
Mathematics Department
Flathead High School
Kalispell, Montana 59901

Lawrence R. Kaber
Mathematics Department
Flathead High School
Kalispell, Montana 59901

Chapter 3

Advanced Technologies as Educational Tools in Science: Concepts, Applications, and Issues

David D. Kumar
College of Education
Florida Atlantic University
Davie, Florida 33314

Philip J. Smith
Department of Industrial and Systems Engineering
The Ohio State University
Columbus, Ohio 43212

Stanley L. Helgeson
National Center for Science Teaching and Learning
The Ohio State University
Columbus, Ohio 43212

Arthur L. White
National Center for Science Teaching and Learning
The Ohio State University
Columbus, Ohio 43212

Chapter 4

Videodisc Technology: Applications for Science Teaching

Derrick R. Lavoie
Department of Biology
University of Northern Iowa
Cedar Falls, Iowa 50614

Chapter 5

Computer Visualization: New Window on Mathematics

David A. Thomas
Department of Mathematical Sciences
Montana State University
Bozeman, Montana 59717

Mark R. Mitchell
Bridger Systems
Bozeman, Montana 59715

Chapter 6

Visualizing Computer Science

Rockford J. Ross
Computer Science Department
Montana State University
Bozeman, Montana 59717

Chapter 7

Getting Started With Supercomputing: An Approach for High School Students

Donald W. Hyatt
Computer Science Teacher & Laboratory Director
Thomas Jefferson High School for Science and Technology
Alexandria, Virginia 22312

Chapter 8

Scientific Visualization in Chemistry, Better Living Through Chemistry, Better Chemistry Through Pictures: Scientific Visualization for Secondary Chemistry Students

Robert Gotwals, Jr.
Education and Human Resources, K-12 Programs Specialist
MCNC Academic Programs
Research Triangle Park, North Carolina 27709

Chapter 9

The National Education Supercomputer Program

Richard Enderton
Mathematics Department
Minnehaha Academy
Minneapolis, Minnesota 55406

Brian Lindow
National Education Supercomputer Program
Lawrence Livermore National Laboratory
Livermore, California 94550

Chapter 10

New Mexico High School Supercomputing Challenge

Marilyn S. Foster
Computing Information & Communications Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Chapter 11

Sharing Multiple Complementary Representations in the Teaching of Science

Nora H. Sabelli
Networking Infrastructure for Education, EHR
National Science Foundation
Arlington, Virginia 22230

Igor S. Livshits
Computing and Communications
National Center for Supercomputing Applications
Champaign, Illinois 61820

Chapter 12

Education and Collaboration in an Evolving Digital Culture

Donna J. Cox
School of Art & Design
University of Illinois at Urbana-Champaign &
National Center for Supercomputing Applications
Champaign, Illinois 61820

Chapter 13

The Hypergraphics Honors Seminar at Illinois

George K. Francis
Department of Mathematics
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Chapter 14

A Syllabus for Scientific Visualization

Alex Pang
Computer and Information Sciences Board
University of California, Santa Cruz
Santa Cruz, California 95064

Chapter 1

The Power of Visualization: The Impact of Graphing Technology on the Secondary Mathematics Curriculum

LEE E. YUNKER

My purpose in this chapter is to provide you with background information on historical developments related to the use of graphing technology (especially the graphing calculator) in the secondary mathematics curriculum in the United States from the early 1980s to the present time. An effort will be made to present a multi-dimensional view including the role of graphing technology and its development, the powerful influences of the professional mathematics organizations, and the present state of mathematics curricula and instruction in the United States.

More specifically, an attempt will be made to provide a comprehensive view of the development and current use of graphing technology in the United States' secondary mathematics curriculum. It is now clear that the use of graphing technology enhances cognitive development through the power of visualization. The ability that exists today to model the algebraic concepts of the curriculum in a visual geometric way has had a profound impact in addressing the differing styles of today's secondary students. In most schools across the U.S. graphing calculators are in the hands of almost every student. For example, over 1,200 students (school enrollment, 1,500) in West Chicago Community High School (a school with a 30% minority population) have graphing calculators and are expected to use them daily.

The graphing calculator is a very powerful tool for exploring, investigating and discovering many abstract mathematical relationships. Today's high school freshmen are graphing as many functions as most of us graphed in all of our secondary and undergraduate work in the field of mathematics. The opportunity for students to have this power is truly revolutionary in relation to the chalkboard, chalk, pencil, and paper of the last 100 years.

The use of graphing technology in the secondary mathematics curriculum is just in its infancy. The spread of its use is being facilitated by the relatively inexpensive cost of acquiring a graphing calculator. In comparison, students frequently spend anywhere from $100-$150 for a pair of brand name athletic shoes for everyday wear. These shoes are often of no use within 4-6 months. In contrast, the cost of the current top-of-the-line graphing calculator is in the neighborhood of $60.

In this chapter, an attempt will be made to provide an inside look at the publishing industry's role as it relates to the use of graphing technology in their educational materials. It will include a look at some of the inhibiting and facilitating factors that are at work in this industry as it operates independently of any governmental rules and regulations, unlike those in several foreign countries we are constantly charged to emulate in mathematics education.

Further, there will be an effort made to assess the present level of professional development of mathematics teachers in the United States with respect to some of the initiatives that have taken place in conjunction with graphing technology. Included will be the impact resulting from the leadership and vision of such professional organizations as NCTM, MSEB, and the MAA.

Finally, a significant portion of this chapter will be devoted to specific examples of current practice. The primary focus will be on the use of graphing calculators or pocket computers (Waits). Examples will come from a variety of areas including Algebra, Geometry, Advanced Algebra, Precalculus, Fractals, Chaos and Dynamics.

Historical Developments

During the 1980s, we had two major concurrent and parallel developments which played a major role in effecting change in mathematics education in the U.S. at the secondary level. First the calculator industry, led by Casio (Japan), developed the first scientific calculator with a graphing display screen; and, second, the National Council of Teachers of Mathematics (NCTM) released their *Curriculum and Evaluation Standards for School Mathematics* (March, 1989).

The *Standards* called for "scientific calculators with graphing capabilities to be made available to all secondary students at all times." In addition, the *Standards* stated that "a computer (should) be available at all times in every classroom for demonstration purposes, and that all students have access to computers for individual and group work." The graphing calculator provided a powerful analytical tool to be interfaced with the reform of the mathematical content of the secondary curriculum. In addition, desktop computers were gaining popularity and were found in many homes; graphing software was also being developed which could be used to extend and enhance their capabilities. As a result, the *Standards* called for increased attention be given to the use of calculators and computers as tools for learning and doing mathematics. Many of the standards at the secondary level include graphing because of the power of visualization to make connections between numerical, algebraic, and geometric representations.

Shortly after Casio's launching of the first graphing calculator, the rest of the industry realized that there was a huge market for such technology in education, and they began to enter the field. Probably the most significant of these entries to date has been that of Texas Instruments, which introduced the TI-81 and TI-85 graphics calculators.

The advancements in this new technology will undoubtedly continue to develop as the availability of more powerful and inexpensive computer chips increases. This appears to be the only limitation being placed on this new graphing technology. The calculator industry constantly weighs expanding the capabilities of the technology against the cost they believe the consumer is willing to bear.

Factors Inhibiting the Use of Graphing Technology

The use of graphing technology at the secondary level is rapidly expanding. This expansion is being spearheaded by the leadership of NCTM with the acceptance of the NCTM *Curriculum and Evaluation Standards for School Mathematics*. Their acceptance is also now being hailed by publishers and testing companies, as well as by the professional mathematics teacher.

One of the major hurdles that most schools have had to overcome is a lack of research regarding the impact of calculator technology on student learning. During the late 1980s, a meta-analysis of research was conducted at the University of Tennessee which indicated that the use of four function and scientific calculators enhanced student learning in almost all grades from first through 8th grade. The only exception was found to be in the grade levels where fractions were first introduced; here the research data was mixed with no clear advantages one way or another. Today, the lack of research is not as acute, particularly at the secondary level and mathematics teachers have accepted the fact that this technology will enhance student learning and performance.

Two other factors that have inhibited widespread usage of graphing technology are staff development and the cost of the technology itself. First, Mathematics teachers need to understand exactly what the appropriate uses are of this new graphing technology and how they might modify their teaching in order to include them in their classrooms. Second is the issue of cost and the belief held by some that it's the school's responsibility to provide this technology.

It's about time we stand up tall and tell it like it is, there are no free lunches out there! Educational opportunities for our children are not, and never have been totally free. Parents must bare some of these expenses directly if they want the best for their children. Those areas in the U.S. where most of the use of graphing technology can be found are where school districts have taken the position that the cost of this technology should be borne by the student.

Professional Development

As with any reform, staff development is absolutely critical. Teachers have to feel comfortable with the changes being proposed and must be in-serviced in terms of their appropriate use. Teachers must acknowledge the changes are necessary and then lead in their local implementation. This is absolutely critical for the changes in technology. Teachers may, as a result, find themselves way out on the cutting edge—a bit scary, but also exhilarating. This may be due in part to two things. First, schools are not necessarily going to be in a materials adoption cycle that would facilitate immediate change; and second, the materials that are adopted (by a school) may not embrace the change to the extent that the professional teacher might desire.

The NCTM has been very concerned with implementation of the *Curriculum and Evaluation Standards* and has now turned its energies toward staff development related efforts. This can be seen through its support of a number of different activities, including new NCTM publications and affiliated group conferences and workshops, specifically related to building an awareness of the individual *Standards* at each grade level. A very relevant case in point is NCTM's efforts regarding the implementation of graphing technology. The NCTM has developed a series of video tapes and a new position statement which strongly advocates the use of calculators at all grade levels, including calculators with graphing capabilities at the secondary level.

Institutions such as Ohio State University, with their C²PC Program directed by Frank Demana and Bert Waits, have had a profound impact on developing a national cadre of teachers who are implementing the use of graphing technology in their classrooms. These teachers are doing staff development in their own local areas and surrounding states. In addition, these teachers are nodes in an outstanding networking structure which encourages the exchanging of teaching ideas on a regular basis.

Waits was one of the members of the writing team for the NCTM *Curriculum and Evaluation Standards*. Internationally, Demana and Waits are viewed as the most prominent in the implementation of graphing technology in the school mathematics curriculum. Demana and Waits have written several textbooks that make extensive use of graphing technology. These materials have found widespread acceptance across the U.S., especially at the precalculus level. These men, along with others, have also had an impact at the college and university level in the U.S. Unfortunately, it must be noted that many mathematics professors and departments at the undergraduate level are much more resistent to any use of graphing technology than their secondary counterparts.

The Publisher's Role

Most curriculum research has shown that the implemented curriculum in U.S. classrooms is dominated by the textbook. It is estimated that 90% of the implemented curriculum is textbook driven. This means that the publishing industry has a profound impact on what actually occurs in our classrooms. For the most part, the publishing industry in the United States is free from government rules and regulations; free to develop whatever materials it thinks are likely to be purchased by schools and teachers. As a result, decisions to be made about curriculum content for these privately held corporations are always motivated by profit. Profit to a publishing company is the primary determining factor when it comes to what appears in a textbook in terms of its content. Content change in a textbook occurs when the publisher views that change to be saleable in the educational community.

It is clear that private industry, motivated solely by the bottom line, will never lead a reform movement in mathematics education. This is the sole responsibility of the professionals in the field. Mathematic education is what "we" make it!

Because of their focus on profit, it is my opinion that the publishing industry has slowed the reform movement in mathematics education in this country. At first publishers were very skeptical about whether or not the *Standards* would be the driving force in leading the curriculum reform in the U.S. Needless to say, the first textbooks that were produced immediately after the release of the *Standards* (1989) reflected only insignificant changes.

As new textbooks were released in the spring of 1991, we saw a more significant step being taken to implement the *Standards*. Today the publishing industry embraces and touts the implementation of the *Standards*. Why such a turn around? Because teachers everywhere are demanding a *Standards*-type product. Again the perceived market dictates the publishing industry's direction. As a result, secondary level textbooks are now assuming the use of graphing technology on the part of every mathematics student.

The Testing Industry: Leading or Trailing?

While the major publishers were dragging their feet on implementing the *Standards*, so to were the major testing companies and services. They were totally ambivalent when it came to making changes relative to the use of any calculators. Their influence regarding the use of technology and the preparation of college preparatory students has been a major inhibiting factor to NCTM's goals of a quality education for every child. Only recently has the testing community begun to change. This is due to the enormous pressure placed on it by NCTM and the acceptance of the *Standards*.

For example, in October, 1990, the College Board of Trustees approved revision of the College Board Admission Test Program ("ATP") including significant change in the Scholastic Aptitude Test ("SAT"). The new ATP "SAT I" will have a mathematical testing component with increased emphasis on critical reading and new student-generated answers in mathematics. SAT-I will be introduced in the spring, 1994. At the same time that these developments are taking place, we're seeing other private testing corporations considering or making changes relative to the use of calculators. Many of them are now re-norming their tests to accommodate the use of calculators.

Graphical Content in Pre-Algebra Mathematics

One of the major concerns regarding student preparation deals with the graphing content of the typical curriculum prior to a student's exposure to graphing at the secondary level. The following observations have been described by Demana, Schoen, and Waits (1991) regarding the current state of affairs in pre-secondary mathematics education:

- In grades 1-6, students have almost no experience constructing a graph of any kind.
- In grades 1-6, variables that are graphed are nominal or sometimes discrete, with variable values placed at equidistant points on the axis, and this continues to be true, to a lesser extent, in grades 7 and 8.
- In grades 1-6, graphs which students have encountered have a limited number of points to be interpreted, with the intervals between those points being meaningless, and this continues to be true, to a lesser extent, in grades 7 and 8.
- In grades 1-6, graphs are almost always presented as existing entities to be interpreted point by point in connection with some "real" context, and no suggestion is made that graphs may be connected to numerical relationships.
- Other than when graphing linear equations and inequalities in grades 7 and 8, students are not expected to make global or qualitative interpretations of graphs.
- Through grade 8, students have no experience with graphs of non-linear functions.

As a result of these findings, it's not surprising that students have many misconceptions about graphs that they bring with them to their secondary mathematics experiences. Therefore, secondary teachers will have to expect that students will need time to make adjustments as they investigate and explore the effects of scaling, global relationships, and the connections among numerical, algebraic and graphical representations of an application or concept.

Recent Research Findings

Demana, Schoen, and Waits (1993) have reported on several recent research projects that have implications for the use of graphing technology in the secondary curriculum. A few of these will be highlighted to emphasize the importance of the use of graphing technology at the secondary level.

Rich (1990) found that students who used the graphing calculator for the entire year of precalculus were far better able to deal with issues of scale on a graph than comparison students in traditional instruction. Rich also found that students who are taught precalculus using a graphing calculator better understand the connections between an algebraic representation and its graph and that they view graphs more globally, and that they understand the importance of a function's domain, the intervals where it increases and decreases, its asymptotic behavior and its end behavior.

Browning (1988) found that high school precalculus students who use graphing calculators for one year exhibited a significantly increased ability to deal with graphing at the more advanced Van Hiele levels of analysis and ordering.

Farrell (1989) also observed that precalculus students who were taught the use of graphing calculators demonstrated greater facility with higher-order thinking skills than traditional students.

Dunham (1990) observed that in college algebra classes requiring graphing calculators, gender-related differences in performance on graphing items were eliminated, while pretest performance on graphing items indicated that females performed at a lower level than males.

Rich also found that precalculus students in traditional instruction made almost no use of graphs outside of units on graphing.

The Power of Graphing Technology

The power of graphing technology to enhance student cognitive knowledge can best be shown through a series of examples.

Let's first consider the equation $x^6 - 9x^4 + 24x^2 - 16 = 0$, and its real roots. This equation would traditionally be solved strictly by numerical and algebraic processes. The Fundamental Theorem of Algebra and the Rational Root Theorem would suggest six roots in the complex numbers with possible rational roots of ± 1, ± 2, ± 4, ± 8, and ± 16. There is no suggestion of multiple roots and certainly no suggestion to graph its related function $f(x) = x^6 - 9x^4 + 24x^2 - 16$. Students in the past rarely made the global connection between the geometric representation of the function and its x-intercepts, and the polynomial equation and its' roots. (This connection, I contend, was only weakly made, if made at all, and then

for only linear and quadratic equations.) Why? The answer should be obvious. How many of us would have wanted to plot a sixth degree polynomial by computing a table of values?

The following sequence of graphs for $f(x) = x^6 - 9x^4 + 24x^2 - 16$ brings to light a powerful visual connection between the function and its graph and the roots to the equation $x^6 - 9x^4 + 24x^2 - 16 = 0$.

Figure 1 may be a surprise to the uninitiated student, but not to one that has had a few experiences with graphing polynomial functions. The viewing window in Figure 1 is not yet displaying a complete graph. Complete graphs are ones which show all their important behaviors (Demana and Waits, 1990). These behaviors include y-intercepts, zeros, relative extrema, and end behavior. Figure 2 is a result of zooming out by a magnification factor of two. It too is not yet a complete graph. However, to the educated eye both Figure 1 and Figure 2 provide a great service in solving the equation. Its clear that this equation has double roots at -2 and 2, with single roots at ±1. These can then be quickly verified by appropriate numerical methods. What else do you think a student might learn from such a geometric representation?

Figures 3 and 4 both display complete graphs of the function $f(x) = x^6 - 9x^4 + 24x^2 - 16$. Students who have little or no experience with the effects of scaling have great difficulty recognizing these two geometrical representations as being of the same function. [Our colleagues in science experience this difficulty on a regular basis.] Regular use of graphing technology has basically eliminated this problem.

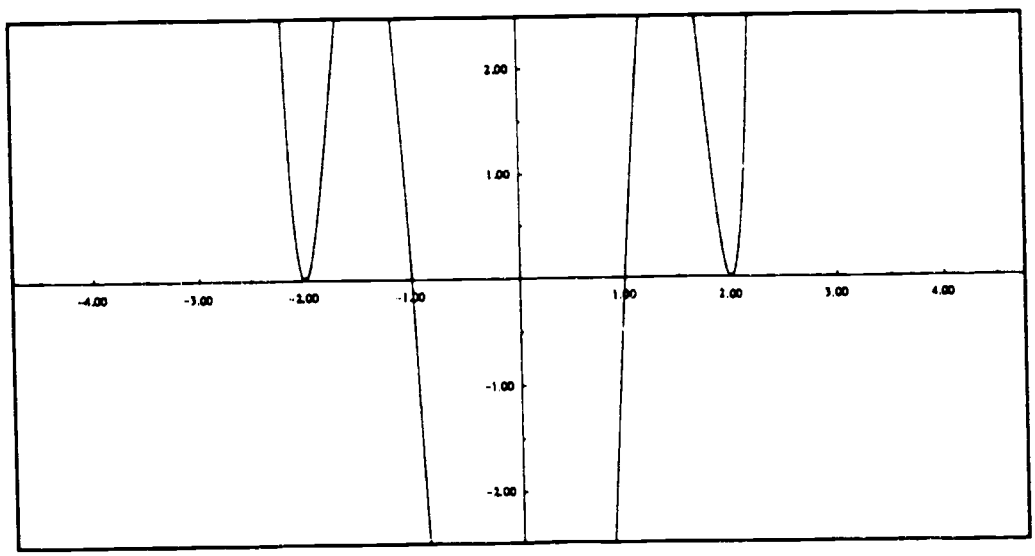


Figure 1. The graph of $f(x) = x^6 - 9x^4 + 24x^2 - 16$ in [-5,5] by [-3,3]

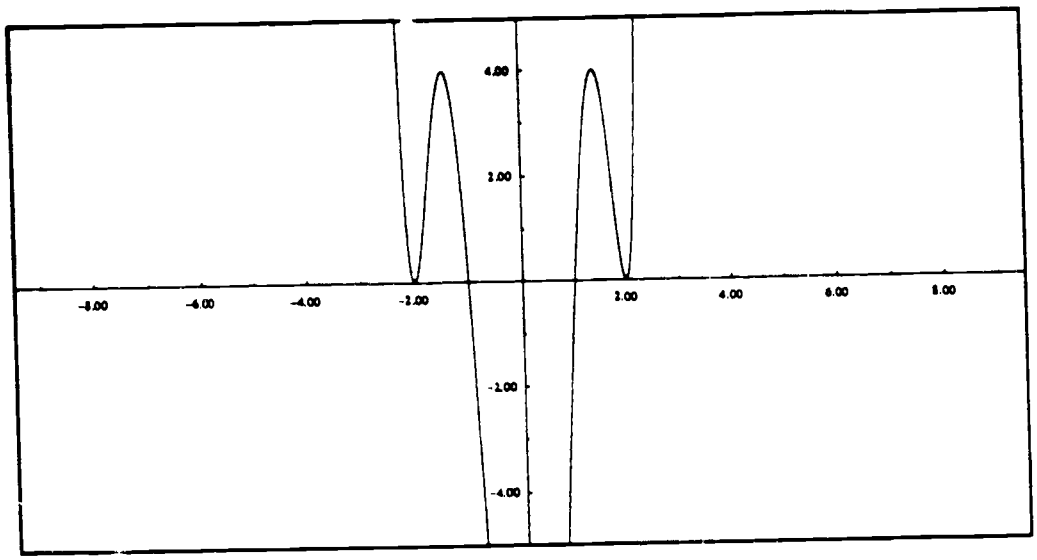


Figure 2. The graph of $f(x) = x^6 - 9x^4 + 24x^2 - 16$ in [-9,9] by [-5,5]

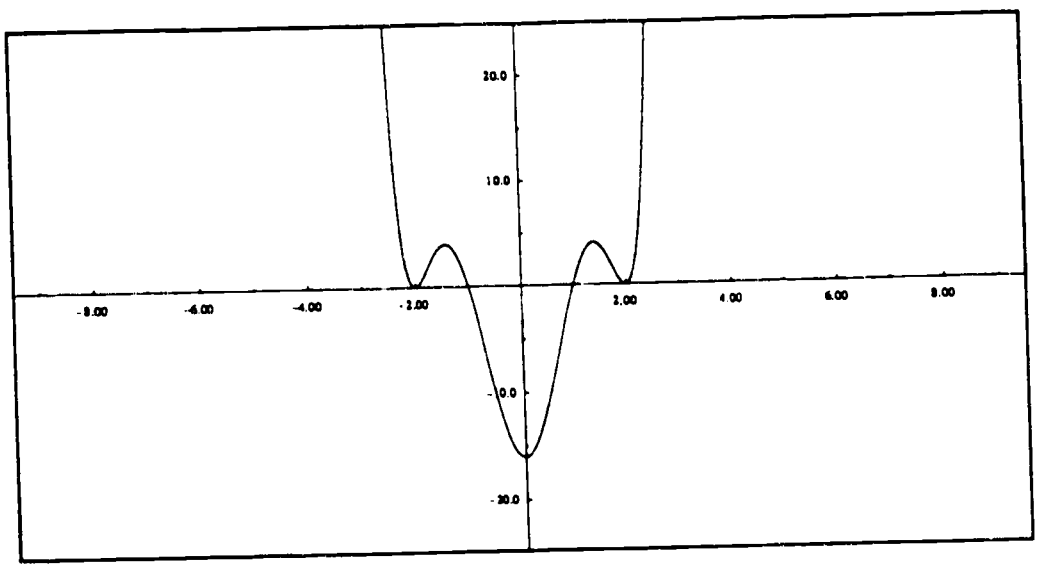


Figure 3. The graph of $f(x) = x^6 - 9x^4 + 24x^2 - 16$ in [-9,9] by [-25,25]

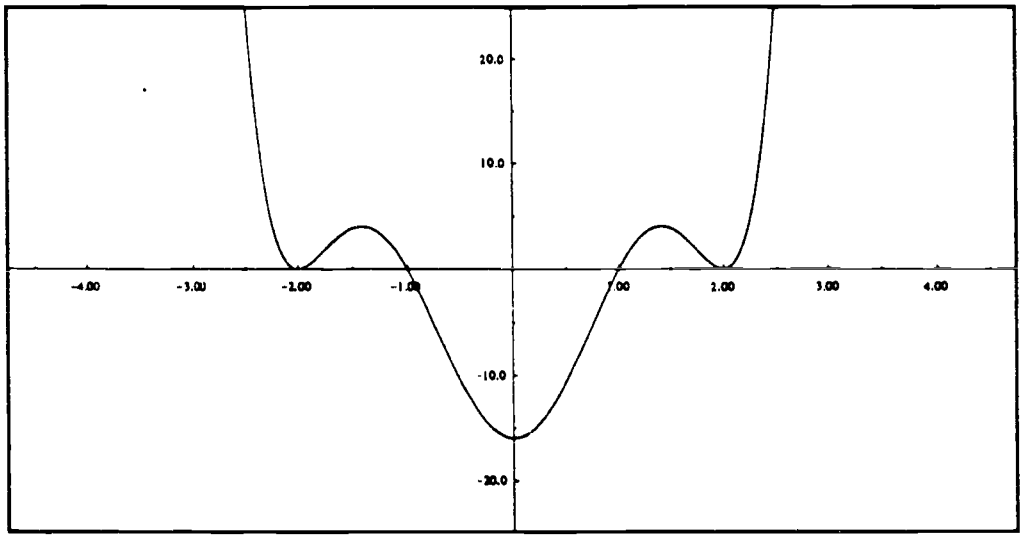


Figure 4. The graph of $f(x) = x^6 - 9x^4 + 24x^2 - 16$ in [-5,5] by [-25,25]

For our second example, let's turn to a purely geometric one. Imagine that you have three vertices of an equilateral triangle labeled T, R, and L for top, left and right respectively. These vertices will be associated with the rolling of a die. To begin, select a random starting point, P_0, and mark it with a dot as shown in Figure 5. Subsequent points, $P_1, P_2, P_3, \ldots, P_n$ are located by placing dots at midpoint locations according to the following algorithm.

- Roll the die.
- Move halfway from the last point marked to the vertex randomly selected by the roll of the die according to the following rule. Move toward T for rolls of 1 and 2, toward L for rolls of 3 and 4, and toward R for rolls of 5 and 6.
- Mark the midpoint by placing a dot at its' location.
- Repeat the process by rolling the die again.

Figures 5-8 show the results of the first three rolls (3, 5, 1, ...) of the die. The first roll of 3 required moving halfway from P_0 toward vertex L and placing a dot at the midpoint, P_1, of segment P_0L. Subsequent midpoints, P_2 and P_3, are shown in Figures 7 and 8. A most significant question is prompted by the execution of this algorithm. What will the final picture containing all points P_n, as n approaches ∞, look like? This question will remain unanswered for the time being to give you an opportunity to reflect on your own visualization power as it relates to this example.

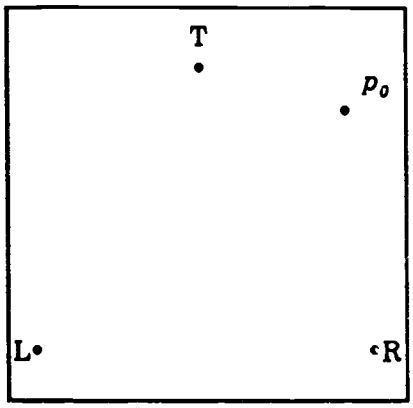


Figure 5. Pick a starting point, P_0

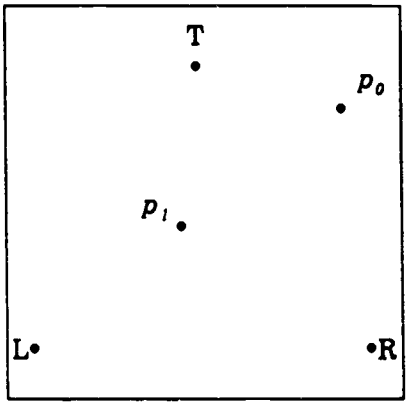


Figure 6. Plot P_1

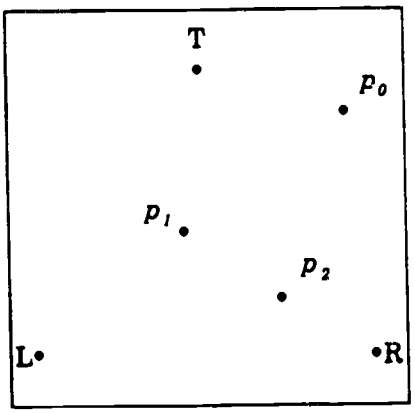


Figure 7. Plot P_2

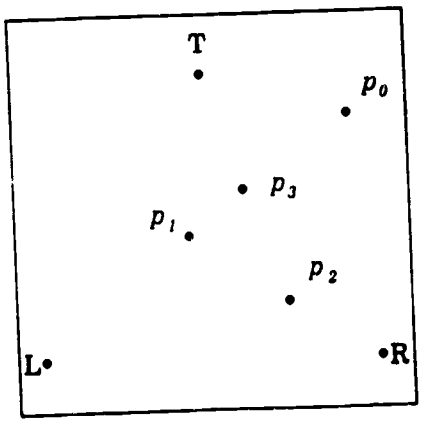


Figure 8. Plot P_3

The third and final example deals with the iteration of the logistics function f(x)= ax(1-x) for initial values in [0, 1]. The process of iteration is the result of doing something over and over again, repeatedly. Iteration is the process of repeatedly forming the composition of a function with itself. This is illustrated in the expresion below in which an argument is repeatedly passed through the same function f(x) in a cyclical fashion.

$$x_0 \rightarrow f(x_0) \rightarrow f[f(x_0)] \rightarrow f\{f[f(x_0)]\} \rightarrow \ldots$$

Another way of thinking about iteration is to consider some initial argument x_0 and obtaining the functional value $f(x_0) = x_1$. Then using x_1 as the next argument and obtaining the next functional value x_2, and so on. The resulting sequence of iterates or arguments are

$$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow \ldots \rightarrow x_n$$

The sequence of iterates shown above is the focus of this final example. We are interested in determining if we can predict the long term behavior of the iteration sequence for different parameter values of a, and also whether or not these patterns are dependent on our choice of the initial value, x_0.

We consider here two different numerical cases. Let's iterate the initial value x_0= 0.20 in two different functions; f(x)= 2.8x(1-x) and f(x)= 3.18x(1-x). For f(x)= 2.8x(1-x), we have the sequence 0.20, 0.44, 0.68, 0.60, 0.67, 0.61, 0.66, 0.62, ... , and for f(x)= 3.18x(1-x), we have 0.20, 0.50, 0.79, 0.52, 0.79, 0.52, 0.79, ... The first iteration sequence appears to be converging, but the exact value is not yet apparent. In the second iteration sequence, we can already see, after only four iterations, that we have period-two cyclical behavior.

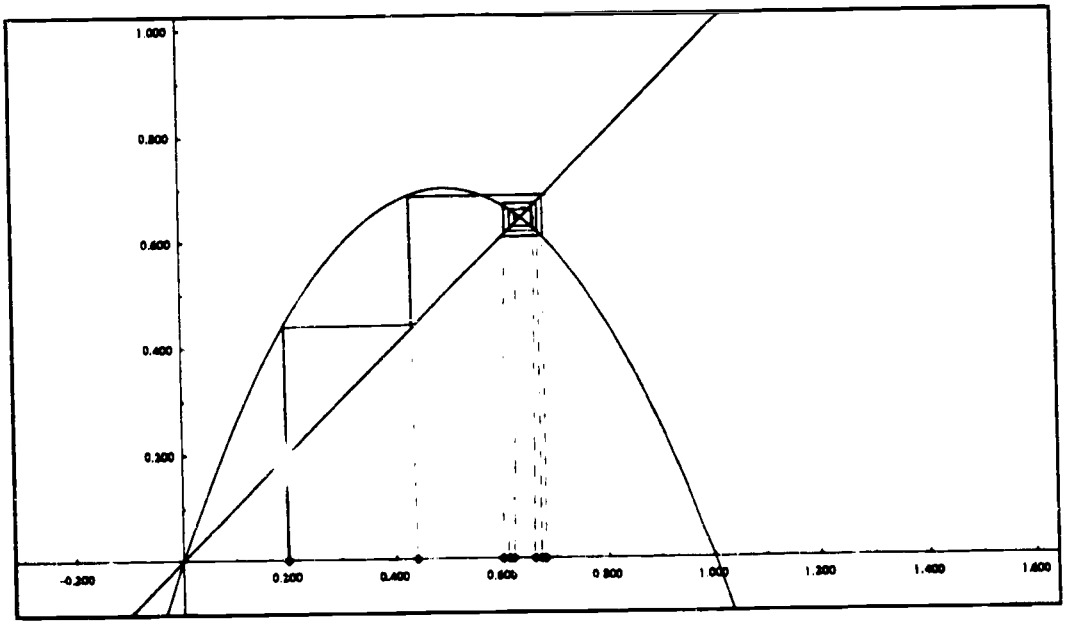


Figure 9. Iterating f(x) = 2.8x(1-x)

Both of the numerical iteration sequences for f(x) = 2.8x(1-x) and f(x)= 3.18x(1-x) can be better understood through the use of graphing technology. Figures 9 and 10 demonstrate the same iterations, but only in a visual sense. The dimension of the viewing window in each case is [-0.5, 1.5] by [-0.1, 1.1].. The iteration patterns in Figures 9 and 10 are marked along the horizontal axis with several "o" to show their converging long term behaviors.

The staircase iteration pattern is also a very powerful visual tool for helping understand the iteration process. When drawing this staircase pattern with vertical and horizontal segments, it is important to always first go vertically from the initial point to the function, then horizontally to the line y = x, then vertically back to the function, then horizontally back to the line, continuing the process repeatedly.

For the function f(x)= 4x(1-x) shown in Figure 11, each reader is asked to carefully perform seven steps of graphical iteration with a pencil and straight edge.

If iteration is performed for an extended number of iterations, the resulting sequence of iterates will be void of any observable pattern. This unpredictable behavior of nonlinear dynamics known as chaos was only first completely described in 1976 by Mitchell J. Feigenbaum, one of the modern fathers of chaos theory. Feigenbaum was the first to explain the phenomena for the transition from stable predictable dynamic behavior to that of chaotic or near-random behavior associated with small changes in a given parameter, such as a in the logistic function mentioned earlier. Today, the discoveries of researchers such as Mitchell J. Feigenbaum, Benoit B. Mandelbrot, Heinz-Otto Peitgen and many others are well within the reach of most secondary students given the appropriate graphing technologies.

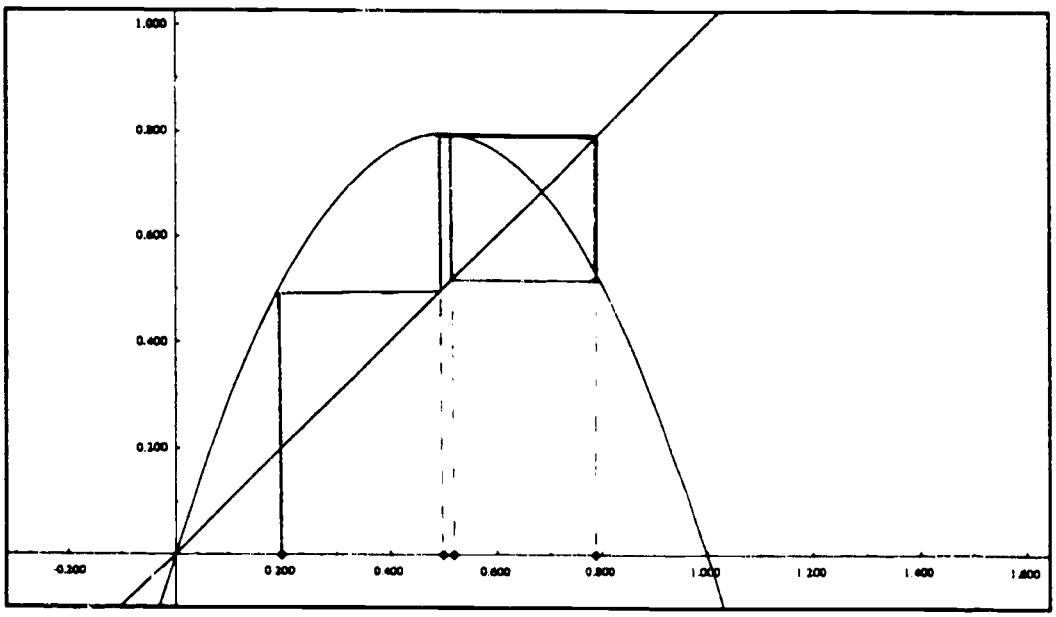


Figure 10. Iterating f(x) = 3.18x(1-x)

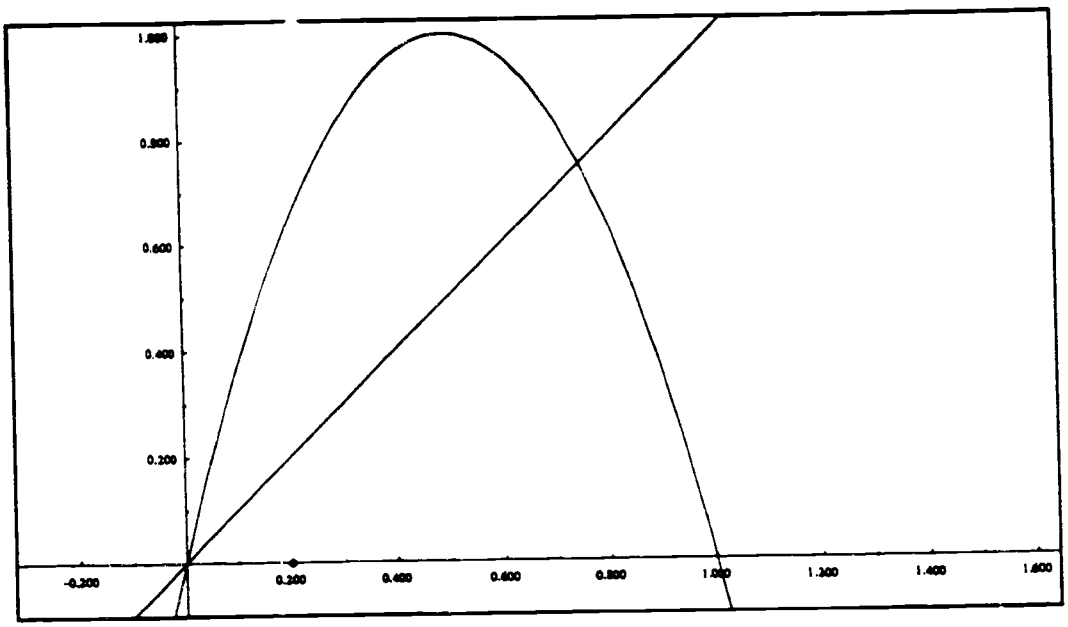


Figure 11. Iterating f(x) = 4x(1-x)

Summary

Today we have all the evidence necessary to vigorously argue that contemporary graphing technologies in the hands of students will enhance their learning. It is the duty and responsibility of each of us to see that every school in our own local neighborhood has the commitment, resources, and trained teachers to take advantage of the opportunities afforded our children with the use of these technologies. I have often been known to ask the question, "What is the school?," and then answered, "Whatever you and I choose to make it!" It's in our hands, and we do make a difference!

Appendix

The first two programs can be used to enhance our understanding of two of the examples presented earlier and to illustrate the power of visualization through the use of graphing technology. The third program, upon execution, produces the beautiful fractal image of a fern through a simple random process using secondary algebra and geometry concepts.

Chaos Game

This program provides an incredibly detailed and highly structured fractal image known as the Sierpinski Triangle, from a purely random process. This beautiful visual model is so very counterintuitive that it challenges our deepest sense of reality. The following key strokes provide the essential steps for programming the Texas Instruments TI-81 graphing calculator.

Select the program mode and enter an appropriate program name such as "The Chaos Game."

Line	Texas Instruments
1	:ClrDraw
*2	:Int (50Rand+1)->Rand
3	:0->Xmin
4	:1->Xmax
5	:1->Xscl
6	:0->Ymin
7	:1->Ymax
8	:0->Yscl
9	:Input
10	:X->A
11	:Y->B
12	:Lbl 1
13	:Int (3Rand)/2->X
14	:0->Y
15	:If X=.5
16	:1->Y
17	:(A+X)/2->A
18	:(B+Y)/2->B
19	:PT-On(A,B)
20	:Goto 1

* The "->" represents the assignment key function performed by pressing the "STO →" key.

Graphical Iteration

This program provides a quick way of iterating the family of quadratic functions whose parent function is the logistics function f(x) = ax(1-x). It allows for a more thorough and extensive exploration of the notions central to an understanding of chaos. Execution of this program requires that the user select the number of decimal places (0 to 10), input the parameter value a, and select the initial value, I, with which to start the iteration process. The following key strokes again provide the essential steps for programming the Texas Instruments TI-81 graphing calculator. Select the program mode and enter an appropriate program name such as "Graphical Iteration."

Line	Texas Instruments
1	:ClrDraw
2	:Disp "DCML PLCS"
3	:Input F
*4	:0->R
5	:0->Xmin
6	:1->Xmax
7	:0->Ymin

```
 8          :1->Ymax
 9          :Disp "A="
10          :Input A
11          :Disp "I="
12          :Input I
13          :DrawF AX(1-X)
14          :DrawF X
15          :AI-All->J
16          :Line(I, 0, I, J)
17          :Goto 2
18          :Lbl 1
19          :AI-All->J
20          :Line(I, I, I, J)
21          :Lbl 2
22          :Line(I, J, J, J)
23          :Pause
24          :IPart ((10^F)*J)/(10^F)->J
25          :Disp J
26          :Pause
27          :R+1->R
28          :J->I
29          :If R>7
30          :End
31          :Goto 1
```

* The "->" represents the assignment key function performed by pressing the "STO →" key.

A Fractal Fern

This program provides another visually incredible fractal pattern. This fractal model has a striking resemblance to the plant world of ferns as we know them. The following key strokes again provide the essential steps for programming the Texas Instruments TI-81 graphing calculator. Select the program mode and enter an appropriate program name such as "A Fractal Fern."

```
Line    Texas Instruments
 1          :ALL-OFF
 2          :ClrHome
 3          :0->Xmin
 4          :100->Xmax
 5          :100->Xscl
 6          :0->Ymin
 7          :100->Ymax
 8          :100->Yscl
 9          :ClrDraw
10          :Disp "HOW MANY POINTS"
```

```
11          :Input N
12          :30->L
13          :64->W
14          :W+L->Q
15          :0.5W->E
16          :0.57W->F
17          :0.408W->G
18          :0.1075W->H
19          :0->R
20          :-0.036W->S
21          :0.0893W->T
22          :0.27W->U
23          :E->W
24          :0->Y
25          :0->I
26          :Lbl 1
27          :Rand ->D
28          :If D>.02
29          :Goto 2
30          :E->A
31          :0.27Y+R->B
32          :Goto 5
33          :Lbl 2
34          :If D>.17
35          :Goto 3
36          :-0.139X+0.263Y+F->A
37          :0.246X+0.224 Y+S->B
38          :Goto 5
39          :Lbl 3
40          :If D>.3
41          :Goto 4
42          :0.17X-0.215Y+G->A
43          :0.222X +0.176Y+T->B
44          :Goto 5
45          :Lbl 4
46          :0.781X+.034Y+H->A
47          :-0.032X+0.739Y+U->B
48          :Lbl 5
49          :PT-On(1.5(A+L)-40,139-1.5(Q-B))
50          :A->X
51          :B->Y
52          :I+1->I
53          :If I<N
54          :Goto 1
55          :End
```

* The "->" represents the assignment key function performed by pressing the "STO →" key.

References

Browning, C. (1988). *Characterizing levels of understanding of functions and their graphs*. Unpublished doctoral dissertation, The Ohic State University, Columbus, Ohio.

Demana, F., & Waits, B. K. (1990). *Precalculus mathematics, a graphing approach*. Reading, MA: Addison-Wesley Publishing Co.

Demana, F., Schoen, H., & Waits, B. (1993). Graphing in the K-12 curriculum: The impact of the graphing calculator. In T. Romberg, E. Fennema, & T. Carpenter (Eds.), *Integrating research on the graphical representation of functions*. Hillsdale, NJ: Lawrence Erlbaum.

Dunham, P. (1990). *Mathematical confidence and performance in technology-enhanced pre-calculus: Gender-related differences*. Unpublished doctoral dissertation, The Ohio State University, Columbus, Ohio.

Farrell, A. (1989). *Teaching and learning behaviors in technology-oriented pre-calculus classrooms*. Unpublished doctoral dissertation, The Ohio State University, Columbus, Ohio.

National Center for Research in Mathematical Sciences Education. (in press). Graphs and functions. Demana, F., Schoen, H.L., & Waits, B. K. *Graphing in the K-12 curriculum: The impact of the graphing calculator*. Madison, WI: University of Wisconsin.

National Council of Teachers of Mathematics. (1989). *Curriculum and evaluation standards for school mathematics*. Reston, VA: NCTM.

Peitgen, Jurgens, & Saupe. (1991). *Fractals for the classroom, part one*. New York: Springer-Verlag.

Peitgen, Jurgens, & Saupe. (1992). *Fractals for the classroom, part two*. New York: Springer-Verlag.

Peitgen, Jurgens, Saupe, Maletsky, Perciante, & Yunker. (1991). *Fractals for the classroom: strategic activities, volume one*. New York: Springer-Verlag.

Peitgen, Jurgens, Saupe, Maletsky, Perciante, & Yunker. (1992). *Fractals for the classroom: Strategic activities, volume two*. New York: Springer-Verlag.

Rich, B. S. (1990). *The effect of the use of graphing calculators on the learning of functions, concepts in pre-calculus mathematics*. Unpublished doctoral dissertation, University of Iowa, Iowa City, Iowa.

Yunker, L.E., Crosswhite, F.J., Elswick, V.A., & Vannatta, G.D. (1991). *Advanced mathematical concepts*. Columbus, OH: Glencoe Publishing Company (Merrill).

Chapter 2

Using Graphing Calculators to Teach High School Mathematics

LARRY KABER
KAREN LONGHART

In 1988, the mathematics department at Flathead High School, Kalispell, MT decided to pilot test the Waits/Demana *Precalculus with a Graphing Approach* text. Because our school district had no money to purchase graphing calculators for the students, we were told that it was not possible to run the test. We then decided that the only way to proceed with the test was to persuade students to purchase their own graphing calculators. As a result, a letter was sent to the students in the affected section, asking them to purchase TI-81 calculators. Instead of the angry phone calls that we expected, parents and students were enthusiastic to try the project.

In the last four years, we have gone from 30 students buying graphing calculators to over 300 students who own them! In cooperation with the Montana Council of Teachers of Mathematics (MCTM), our department now acts as a purchasing agent for the State of Montana so that we can buy the calculators in large quantities and get the most competitive prices possible for our students. This approach frees up money from budget tight school districts where it would be nearly impossible to ever acquire enough money to buy a reasonable number of graphing calculators. So don't wait for your school district to come up with the funds to purchase this affordable technology. Ask your students and their parents to support your effort to enhance their children's mathematics education. The response in our school has been fantastic! Students are enjoying their mathematics classes more and they are anxiously awaiting the arrival of the TI-85. There was even an article in the school newspaper giving the specifications on the TI-85!

For the students who can't afford to purchase their own calculator (there are surprisingly few) we have purchased ten TI-81's for our library and the students can check them out. Also, we have gone to local service clubs (Lions and Kiwanis, for example) and asked them to sponsor needy students.

Transformational Graphing

Encouraging students to purchase their own graphing calculators has many advantages. When a student owns his/her own calculator, exploration and discovery can easily take place at school, at home, or even when riding the school bus. Like any high-tech device, students tend to spend more time using it if they own it.

One of our objectives is that students learn the graphs of many "Parent Functions." For example, by the end of fourth year of mathematics, we expect our students to know "intimately" what the graphs of

$y = x$
$y = x^2$
$y = x^3$
$y = x^n$
$y = 1/x$
$y = a^x$
$y = \sin x$
$y = \cos x$
$y = \tan x$
$y = |x|$
$y = \lfloor x \rfloor$ and
$y = \log_b x$

look like. They also learn how the graphs can be transformed or manipulated with numbers. For instance, they learn the transformations of $y = A*f(B(x - C))+D$ so that they can create any graph that they wish. These transformations are general and apply to any function. Their properties are as follows:

Transformations

A: $|A| > 1$ produces a vertical stretch by a factor of A
$|A| < 1$ produces a vertical shrink by a factor of A
$A < 0$ produces a reflection over the x-axis
B: $|B| > 1$ produces a horizontal shrink by a factor of 1/B
$|B| < 1$ produces a horizontal stretch by a factor of 1/B
$B < 0$ produces a reflection over the y-axis
C: produces a horizontal shift C units
D: produces a vertical shift D units

The order of transformations is to first perform vertical or horizontal stretch/shrinks and/or reflections, followed by vertical or horizontal shifts.

Once the students understand how to graph parent functions and how to perform transformations, they can graph almost any function encountered in high school algebra and introductory calculus. For example, to graph $y = -2 * \text{Log}(-(x-3))+1$, the student would proceed as follows:

1. Recognize that the parent function is $y = \text{Log } x$ and sketch the graph. Notice that the graphing utility does not produce a correct graph of this function as it does not show any asymptotic behavior. This asymptotic behavior occurs in many different func-

tions, which further illustrates why students still need to know what the parent function looks like. They also need to realize that discrete graphing utilities produce visual approximations rather than exact graphs.

2. Reflect the graph over the x - axis and introduce a vertical stretch by a factor of 2 to produce y = -2 * Log x.
3. Reflect the graph over the y axis to produce y = -2*Log (-x).
4. Shift the graph left three units to produce y = -2*Log (-(x - 3).
5. Shift the graph up one unit to produce the final graph.

With instruction and practice in these procedures, students develop a strong sense of how to graph by the use of parent functions and transformations. Along with this sense of graphing comes an understanding of domain and range. Data analysis and curve fitting are a natural extension of this understanding.

Sequences and Series Graphically

Many students have trouble understanding the relationship between arithmetic and geometric sequences and series. They also fail to make the connection between arithmetic sequences and linear functions, and geometric sequences and exponential functions. Using graphs and the following program, these problems can be cleared up.

```
PRGM1: SEQUENCES
:1 -> Xmin
:96 -> Xmax
:10 -> Xscl
:-100 -> Ymin
:100 -> Ymax
:10 -> Yscl
: Disp "ARITHMETIC (0) OR GEOMETRIC (1)"
:Input A
:If A = 0
:Goto 1
:Disp "FIRST TERM"
:Input T
:Disp "COMMON RATIO"
:Input R
:"T * R ^ (X - 1)" -> Y1
:"T *(1 - R^ X)/ (1 - R)" -> Y2
:Dispgraph
:End
:Lbl 1
:Disp "FIRST TERM"
:Input T
:Disp "COMMON DIFFERENCE"
:Input D
:"D(X - 1) + T" -> Y1
```

```
:"(X/2)*(2T + D(X - 1))" -> Y2
:Dispgraph
:End
```

This program will graph any arithmetic or geometric sequence and its series. The student can trace on the graph and find any of the first 96 terms of a sequence or the sum of a series. One example that shows how this program illustrates arithmetic sequences and series graphically produces the arithmetic sequence -8, -7.5, -7,... by letting T = -8 and D = 0.5. The students can easily see the linear function that describes the sequence and the quadratic function representing the series. To illustrate geometric sequences and series, produce the geometric sequence of 1, 2, 4, 8,... by letting T = 1 and R = 2. Next, let T = 40 and R = 0.5 (ie., 40, 20, 10,...). This demonstrates the concept of what a convergent geometric series means graphically.

Data Analysis

Using the graphing calculators every day in class helps students become "graphically literate" and helps them develop a thorough understanding of functions and their behavior. Data analysis is helpful in tying all of these concepts together. Two examples follow where students can model problems with linear, exponential or logarithmic regression curves. In these problems, we require the students to decide which regression curve bests fits the data, and why.

Task #1: Some researchers believe that women runners might start beating men in world-class competition within a few generations. Are the researchers correct? In the table below are the winning times for the men's and women's 200 meter track and field event at the Olympics from 1960-1988.

200 meters

Year	Men	Women
1960	20.5	24.0
1964	20.3	23.0
1968	19.8	22.5
1972	20.0	22.4
1976	20.2	22.3
1980	20.2	22.0
1984	19.8	21.9
1988	19.7	21.3

Analyze this data using either a linear, exponential or logarithmic regression equation and make a prediction as to whether women will actually ever catch up to men in the 200 meter race. After you have analyzed this data mathematically, state reasons why you do (or do not) support this analysis.

Using a graphing calculator and its statistical features, a pair of regression equations can be found. If this data is modeled with logarithmic regression curves and then graphed, it can be shown mathematically that there appears to be a time when indeed women will catch and surpass men in this event. This problem brings up an issue that can be debated by scientists concerning the physiology of men versus women. A third issue that we discuss is the danger of extrapolating too far into the future from the data. This problem is well received by students because of their interest in sports.

Task #2: What year was it 1 million seconds ago? What year was it a billion seconds ago? What year was it a trillion seconds ago? Surprising? With your new found appreciation for the magnitude of a trillion, look at the U.S. federal debt.

Year	Federal Debt
1980	909,000,000,000
1981	994,000,000,000
1982	1,100,000,000,000
1983	1,400,000,000,000
1984	1,600,000,000,000
1985	1,800,000,000,000
1986	2,100,000,000,000
1987	2,300,000,000,000
1988	2,600,000,000,000
1989	2,900,000,000,000
1990	3,200,000,000,000
1991	3,600,000,000,000

Because the amount of the federal debt depends on the year, let x equal the number of years after 1980 and let y equal the dollar amount of the federal debt. Use the statistical capabilities of your graphing calculator and calculate the linear regression and exponential regression curves for this data. Graph the two curves and use the graphs to approximate the federal debt in 2000. What is the difference in the two forecasts? Which one do you think is more accurate? In the linear regression equation, what does the slope of the function represent in the real world? In the exponential regression equation, what does the base of the exponent represent in the real world? This problem can lead to further discussions in a economics class or a history class as to what the implications are for such a high national debt.

Graphing Calculators and Calculus

Topics from calculus which can be enhanced through visualization include: end behavior of a function, limits at a point, critical values of a function, and numeric integration procedures, to name a few. The purpose of some of the topics of traditional calculus is to provide an analytic and algebraic approach for discussing functions. Maybe the emphasis on finding critical values algebraically is not needed, especially when technology can be

used for the same purpose. The mathematical modeling of functions is much easier to handle with graphing utilities. To understand that most function models are piece-wise functions means that we need to have a clear understanding of parent functions and their transformations.

Visualizations of the end behaviors of various functions provide students with an intuitive basis for the concept of a limit, especially as the independent variable becomes very large, or very small. This visual support adds validity to the abstract approach generally used in a traditional calculus course, which typically involves the use of differentiation. Furthermore, divergent and convergent sequences can now be discussed very intuitively.

The relationships between functions and their inverses are much easier to discuss with visual support, especially in the parametric form. For example consider $y = e^{(-x)^2}$ with a domain of $[0,\infty)$ and the inverse function $f^{-1}(x) = \sqrt{(-\ln x)}$. This latter function will bother most students because it involves taking the square root of a negative number; but as soon as you make a visual representation of the inverse function, it becomes very apparent that $(-\ln x)$, for a domain of (0, 1] is a positive real number.

Investigating inequalities using a graphing utility gives students another tool to use. Consider the following inequalities:

$$x^2+4x \geq 4x, \quad (2x^2-3x-20)/(x+3) < 0, \quad \sin^2 x+\cos 2x \geq 2X.$$

Graphically, the solutions are found with great ease and visual clarity. For example, to solve the first inequality with your graphing utility, simply input each side of the inequality in the form of $Y1 = x^2+4x$ and $Y2 = 4x$. The visual solution to the second inequality is done by graphing the rational function $Y1 = (2x^2-3x-20)/(x+3)$. The third problem would be best solved by letting $Y1 = \sin 2x+\cos 2x-2x$ and then using the trace key to find the zeros. Using the trace key and considering the error, you can produce accurate solutions to many problems of this type.

In the case of a transcendental function combined with an algebraic function, the solution might be attainable only through the use of a graphing utility. An example would be to find the solution of $f(x) = g(x)$ where $f(x) = x^2$ and $g(x) = 2^x$. If you graph both of these functions you might miss at least one of the solutions. However, if you graph $h(x)=x^2-2^x$, it becomes obvious, with a correct choice of range that there will be three solutions.

New Tools — New Teaching

The teaching and learning of mathematics for secondary schools has once again taken a giant step. In this chapter we have presented some uses of visual representation in data analysis. However, graphing calculators can affect all of the topics of a typical secondary school mathematics curriculum. Two caveats are, (1) not all students learn in the same way, and (2) as teachers we teach as we were taught. The power of visualization has allowed mathematics students and teachers to address both caveats. As a student you have another avenue to follow in interpreting mathematical information and making predictions. As a teacher you have a new and creative way to present information. AP-Calculus curricular materials have been static for many years, however the process of teaching has become very dynamic. The realization that all secondary school mathematics can be taught using manipulatives has made calculus easier to comprehend for some students. We consider the

visual approach to be manipulative. We need to consider the use of electronic visual representation as a new delivery system for this subject. The graphing calculator can be a manipulative tool for many of our courses.

During the past decade it has been a goal in many high schools for all mathematics classrooms to be equipped with a computer and monitor or projection device. Once that goal was met, it had to be adjusted because of newer technology, namely the graphing calculator. It is now just as important for us to have overhead graphing calculators as computers. A properly equipped mathematics classroom should have both. The power of visualization along with algebraic models allows more students access to this and practical mathematical topics. For example, more students fail calculus because of poor algebra skills than for any other reason. With the use of visualization maybe more students can find success in the study of calculus. We would hope that college level educators are also adjusting to these new technologies.

Conclusion

As with other secondary school mathematics topics, the power of visualization allows students to spend more of their time on problem solving and less on mechanical manipulations. A majority of elementary calculus is spent on functional analysis so that a visual model can be produced. With the new technologies this time might be spent on applying the calculus to real world problems. Furthermore, the support given by visual representation to algebraic processes brings a whole new dimension to mathematical thinking and modeling.

Chapter 3

Advanced Technologies as Educational Tools in Science: Concepts, Applications, and Issues

David D. Kumar
Philip J. Smith
Stanley L. Helgeson
Arthur L. White

Advances in conceptual approaches, as well as in software and hardware technologies, offer powerful methods for enhancing the use of computers as educational tools in science. The use of hypermedia to provide non-linear access to text, graphics, sound and video is one such important advance (Conklin, 1987; Halasz, 1988; Glusko, 1989; Norman, 1988). The incorporation of "intelligence" in a tutoring system is another advance (Anderson, Boyle & Reiser, 1985; Clancey, 1984; Kearsley, 1987; Sleeman & Brown, 1982; Wenger, 1987; Woolf & McDonald, 1985). While there have been a variety of efforts to make use of these advances, there are many important, unanswered questions that need to be dealt with in order to assess the effectiveness of these technologies and to guide the design of effective tutoring environments.

This chapter describes the concepts, applications and issues associated with two rather different technologies, the use of hypermedia [including level-3 interactive video (IVD)] and intelligent tutoring systems, in science education. One approach [as demonstrated by the "Gait Analysis Instruction Tool (GAIT)," "Hyperequation," and "Hyperscience 456"] involves using hypermedia techniques (including IVD) to provide instruction, to assess the process of problem solving, and to provide a context for problem solving respectively. A second approach (as illustrated by the "Transfusion Medicine Tutor" later in this chapter) provides a problem-based learning environment (Barrows, 1988) in which intelligent tutoring capabilities are incorporated to provide feedback and guidance to the students.

Hypermedia and Level-3 Interactive Video

In 1945 President Roosevelt's science advisor, Vannevar Bush, wrote in the *Atlantic*, describing a (hypothetical) tool that would link related pieces of information. Such a tool could be used to manage information in new and innovative ways, by forming omni-dimensional associations or links (Tsai, 1988; Marsh & Kumar, 1992). Bush has been credited with being the pioneer of this idea of "using a machine to store connections between pieces of information" (Smith, 1988, p. 33).

Hypermedia is based on this idea of linking related information. It is an interesting extension in that very different types of information and information displays are linked, ranging from text and simple graphics to video. According to Halasz (1988), hypermedia represents "a style of building systems for information representation and management around a network of multi-media nodes connected together by typed links" (p. 836). The design of a hypermedia environment is supported by software such as the "HyperCard (TM)" and "SuperCard (TM)," which allow the creation of networks of interconnected electronic cards, or screens, to represent a collection of related ideas in the form of visual text and graphics, and to facilitate the organization, storage, and retrieval of information (Halasz, Moran & Trigg, 1987; Halasz, 1988). In such environments, each screen is thought of as a "notecard" (node) and the associated concepts are linked via electronic "buttons" (links) (Dede, 1987; Halasz, 1988).

In addition to linking each card with additional printed information, links can also be made to nodes containing "information" such as audio or video (Mulhauser, 1992; Ambron & Hooper, 1988; Aambo & Hovig, 1988). For example, in level-3 interactive video systems, software such as HyperCard (TM) in an external microcomputer is used to control the learner-video interaction allowing "students to manipulate audiovisual materials stored on the videodisc in numerous ways" (Litchfield & Dempsey, 1992, p. 40). A reasonable amount of educational applications of hypermedia are in the level-3 interactive domain.

Proponents of hypermedia suggest that it can be used effectively to support learning for a number of reasons:

1. The use of audio and video displays make it possible to provide richer environments in which information is provided in real-world contexts (Kumar, 1991a; Hofwolt, Kumar & Altman, 1991; Litchfield & Dempsey, 1992). Such contexts arguably serve to motivate students, as well as enhance recall of important points.
2. The use of links to easily access related knowledge encourages students to explore these relationships. The assumption is that "the more links that can be formed between existing knowledge and new knowledge, the better the information will be comprehended and the easier learning will be" (Jonassen, 1988, p. 13).

Intelligent Tutoring Systems

Over the last two decades, there has been a great deal of interest in the development of intelligent tutoring systems (Anderson, Boyle & Reiser, 1985; Clancey, 1984; Sleeman & Brown, 1982; Wenger, 1987). The assumption behind this research has been that, with greater "intelligence," computer systems can provide more adaptive and therefore more effective tutoring. According to Woolf (1987) the goals of intelligent tutoring systems

include representing knowledge, monitoring student learning, tailoring instruction to the individual learning needs of students and providing a macro context for learning.

The goal of this research, then, has been to build highly adaptive teaching machines. These computer systems not only have knowledge of how experts perform problem solving, but also have other knowledge relevant to teaching. Included is knowledge of students' common "naive conceptions" and errors. Also included is knowledge about how and when to apply various teaching strategies. Some intelligent tutoring systems (ITS) design concepts are described below.

ITSs typically have three major components: The "expert system," the "student model" and the "tutor." The knowledge provided by these components is used in a variety of ways to support interactive teaching.

The expert system provides a representation of the knowledge and problem solving processes consistent with correct expert performance. This knowledge is used for two purposes: To help detect errors on the part of a student, and to provide supporting explanations and teaching about correct performance. One function of this expert knowledge, then, is to help the system develop its "student model" for a given student (a representation of what the computer thinks the student does and does not know). Students may differ from the expert model in that they may be missing some of the declarative or procedural knowledge important for expert performance, or they may have incorrect or naive declarative or procedural knowledge. Another function of the expert module is to support the tutoring module in providing explanations and guidance to the student.

The student model for a given user (student) is developed by observing the correct and erroneous performances of the student. To make inferences from observed behaviors, the computer makes use not only of its expert knowledge (to answer the question: Does this student's performance differ from that of an expert?), but, also, of a collection of knowledge about stereotypical incomplete or naive conceptions that students often have. Thus, like an expert human tutor, the ITS knows what types of naive conceptions students typically have, and can infer their existence from the errors a student is making.

Tutoring involves more than determining what naive conceptions and areas of ignorance a student has. Thus, the "tutor" must make use of the knowledge provided by the "expert system" and also of the insights provided by its "student model." The "tutor" must implicitly or explicitly consider alternative teaching methods based on the current context (and its interactions with the student up to that point). In discussing the design of an ITS, Collins, Warnock and Passafiume (1975) suggest a number of teaching principles, such as:

1. Asking the student to parrot what she/he has just read is "a mode of recall that leads to little or no long-term retention" (p. 70);
2. Asking review questions covering previously taught material when it comes up in another context is an effective way to reinforce learning;
3. Asking a question about the student's wrong answer and not simply teaching the student the right answer when he/she makes a mistake helps "the student remember the distinction" (p. 73).

Similarly, Woolf (1987) suggests selecting learning tasks which "illustrate similarities among related phenomena" (p. 232) and which provide "heuristic knowledge" (p. 239). Thus, the literature on ITS research provides a great deal of insight into the design of educational systems, both at an architectural level and in terms of principles for effective teaching.

Sample Systems and Research Issues

In order to highlight important research questions associated with the use of hypermedia (including level 3 interactive video) and ITS technologies, specific applications are reviewed below.

Hypermedia Systems

The Gait Analysis Instructional Tool (GAIT) is an example of a hypermedia tutoring system designed to teach aspects of orthopedics with case studies of real patients. This system provides a problem solving environment in which students learn while answering questions and analyzing complete patient cases (Barrows, 1988; Boring & Nutter, 1984; Burnett, Mahoney, Chidley & Pierson, 1986; Cardiff, 1986; Irby, 1986; Henry, 1985). GAIT has two components. The first, written in SuperCard (TM) asks students questions and provides access to text and video to help them learn. The second, written in C, provides access to complete patient cases. GAIT runs on a Mac II using three color monitors. When solving complete patient cases, students can request any of the data normally available to a physician by clicking on the corresponding button (see Figure 1). The set of available data is always displayed on the left screen.

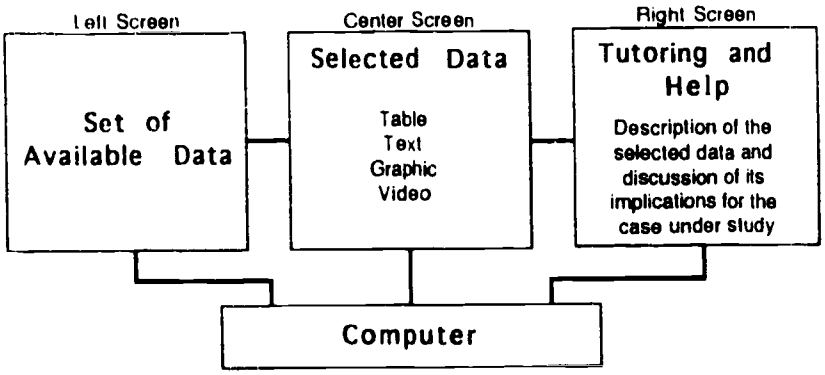


Figure 1. Schematic diagram of GAIT display

When a particular piece of data is selected for viewing, it is displayed on the center screen (see Figure 1). The data displayed may be a table (text), graphics or video. The student can use the available data to make inferences (ruling out certain hypothesized dysfunctions). If the student has problems doing so, he/she can click on any data display with the mouse and a help window appears. This help window includes written text describing this datum in general, as well as providing a discussion of its implications for the current case. Such full patient cases are suitable for more advanced students.

For novices, GAIT provides access to an on-line "book" (complete with table of contents, index, and glossary) and a set of specific questions that students should be able to answer. By clicking on an entry in the glossary, for instance, a definition appears in a pop-up window. Specific sections identified in the table of contents also have associated graphics and video, which can be displayed upon accessing that section of the "book." Figure 2 shows this component of the system.

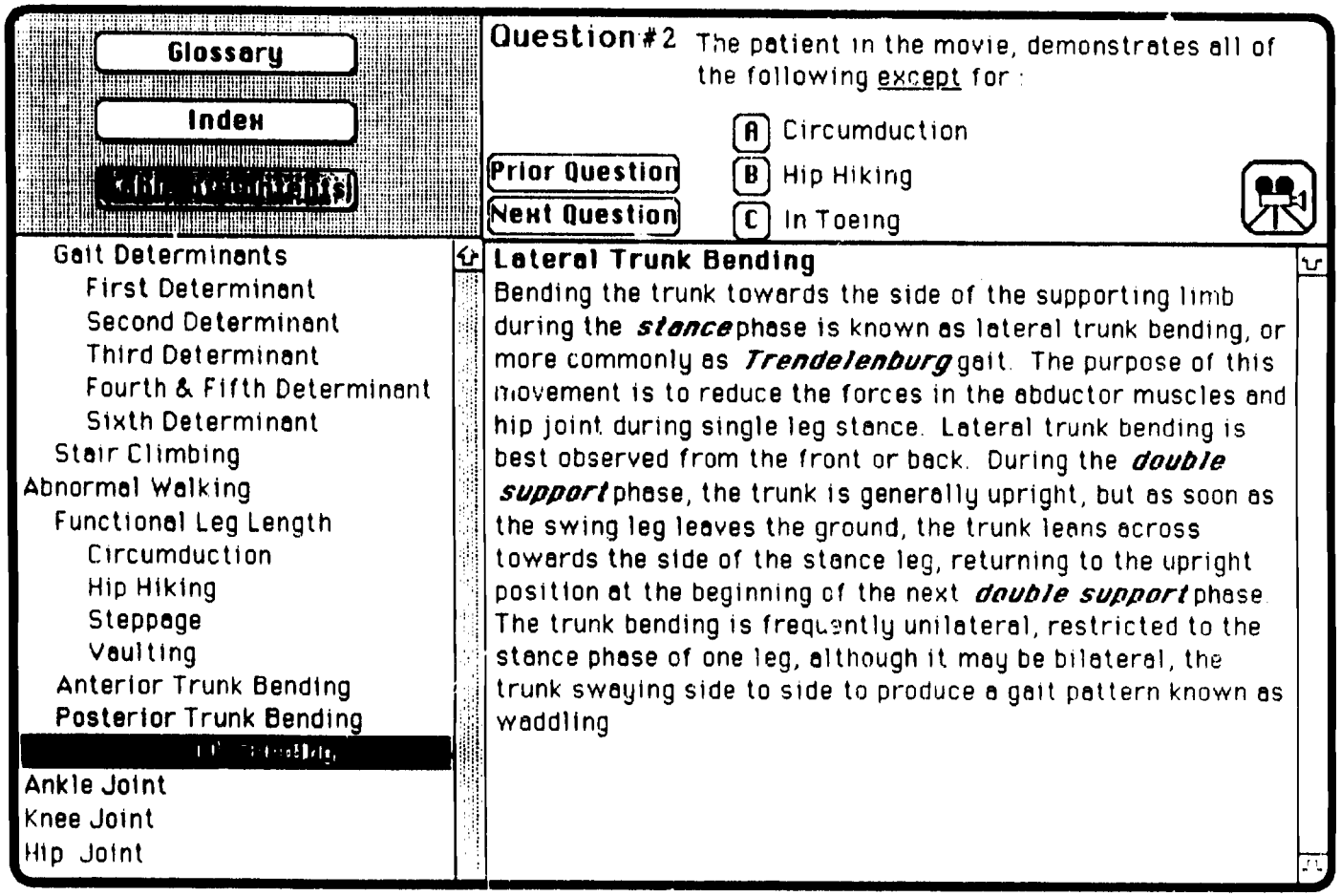


Figure 2. Sample GAIT on-line "book"

A second way to use this component of GAIT is to try to answer the study questions. When the student answers a question, GAIT provides feedback and tutoring. This tutoring provides access to:

1. A context-sensitive table of contents that indicates the relevant sections of the on-line "book" to read;
2. A video display in which an expert discusses the answer to that question (audio feedback) while appropriate video (of a patient, for example) and graphics are displayed.

In general terms, GAIT has three interesting features. First, information is displayed in several different media (text, graphics, speech, and video). Second, this information is linked as appropriate to provide easy traversal among related pieces of information. Third, there are different conceptual approaches to accessing the information (browsing through an on-line "book" vs. asking for tutoring in response to a particular problem). These three features serve to illustrate the capabilities provided by hypermedia.

An interesting, yet challenging, application of hypermedia is in the design of assessment or evaluation tools. According to Shavelson, Baxter, Pine, Yure, Goldman, and Smith (1990), standardized paper-pencil tests are not sufficient to measure the process skills involved in hands-on science instruction.

Simulations of hands-on problem solving while using computers to teach often serve to develop higher order cognitive skills (Gilman & Brantly, 1988). But, such computer based learning often lacks assessment systems that are "computer gradable" (Moore, 1989).

One potential solution to these problems is the use of hypermedia systems that are capable of assessing the process of learning. Under the New Technologies Focus Area at the National Center for Science Teaching and Learning, research on alternative assessment systems using HyperCard (in a Mac II computer) is in progress. Using custom developed assessment software called "Hyperequation" (Kumar, 1991b), the performance of high school students in the task of solving stoichiometric chemical equations has been studied.

The term Hyperequation refers to an approach to writing and balancing chemistry equations using HyperCard in Macintosh computers. Some of the features of Hyperequation include easy operation using the computer mouse, immediate feedback, and the ability to register information pertaining to the process of problem solving such as the order of responses made by the students. In addition, Hyperequation also provides an item by item score for each student. In effect, Hyperequation not only substitutes for a paper-pencil test involving balancing chemical equations, but also provides a non-linear visual assessment environment on a computer screen. A sample Hyperequation is shown in Figure 3. See Figure 4 for a sample Hyperequation record storage.

Other advantages of Hyperequation include the following. Hyperequation is easy to write using HyperCard. It can be linked to databases of chemical indexes, electronic configurations, chemical bonding, and HyperCard periodic tables in case the student wants a quick review of some background information. As HyperCard software, Hyperequation can be linked to selected video segments from professionally produced chemistry videos (e.g., the "Periodic Table Videodisc" of Project Seraphim) through electronic buttons and transformed into a tool for instruction in chemistry. Developments are underway to refine Hyperequation to incorporate capabilities that would recognize and react to the problem solving strategies employed by students.

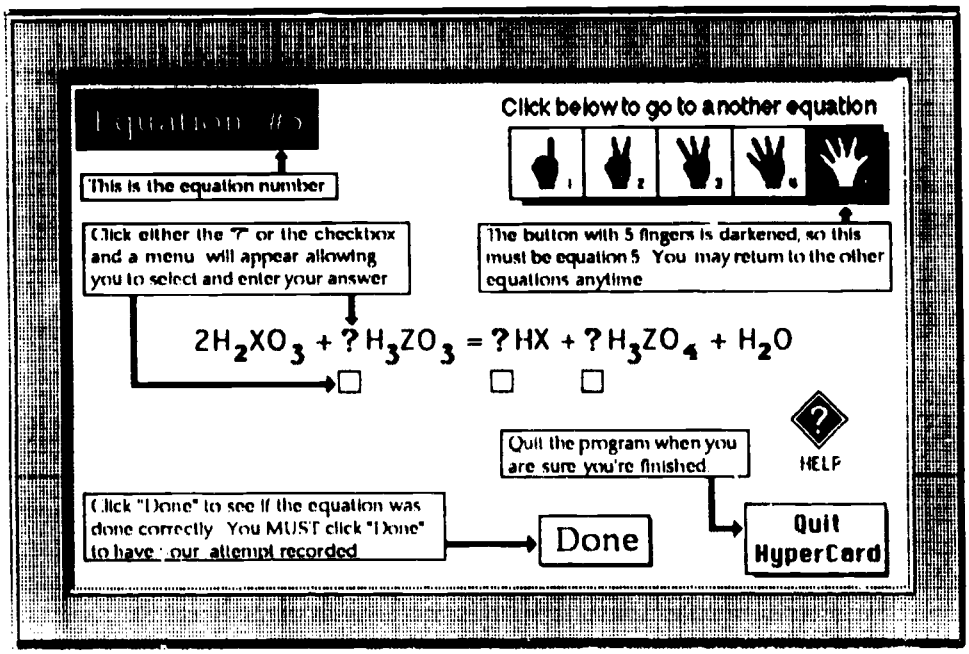


Figure 3. Sample Hyperequation *(Copyright 1991 David D. Kumar)*

Name: Student 3 **Time In:** 2:23 PM **Total Time:** 13.7

Date: 10/24/91 **Time Out:** 2:37 PM

	Attempted	Solved	Errors
1. $H_2SO_4 + 2NaOH = Na_2SO_4 + \underset{1}{\overset{2}{2}}H_2O$	☒	☒	1
2. $\underset{2}{\overset{2}{2}}HCl + Na_2CO_3 = \underset{2}{\overset{8}{2}}NaCl + H_2O + CO_2$	☒	☒	1
3. $2Fe(OH)_3 + 3H_2SO_4 = Fe_2(SO_4)_3 + 6H_2O$	☐	☐	
4. $\underset{2}{2}H_3PO_4 + \underset{3}{3}CaSO_3 = Ca_3(PO_4)_2 + \underset{3}{3}SO_2 + \underset{3}{3}H_2O$	☒	☒	0
5. $2H_2XO_3 + \underset{1}{\overset{2}{5}}H_3ZO_3 = \underset{2}{\overset{2}{2}}HX + \underset{2}{\overset{1}{5}}H_3ZO_4 + H_2O$	☒	☐	3

3 of 9

TOTAL ERRORS 5

Figure 4. A sample display of Hyperequation Report *(Copyright 1991 David D. Kumar)*

Another application of hypermedia in assessment is evident in an on-going project reported by Martinez (1991) where an "IBM-compatible computer interface delivery" platform has been used for the delivery of figural response assessment items in cell and molecular biology. Martinez (1991) has used a "figural response item format" in a computer environment which enables the measurement of knowledge that is difficult to express in verbal or numerical forms. Using a set of computer screen tools activated by buttons (e.g., "move object," "rotate," "draw line"), chromosomes and molecular groups are moved on the screen by students to respond to various questions. One such question reads as follows: "Given the D-glucose below, construct its L-glucose stereo-isomer using the template shown." Martinez (1991) concluded that figural response assessment strategies, in combination with existing assessment methods, "broaden the kinds of thinking called for by tests." Similar work in physics at the University of California-Santa Barbara in collaboration with the California Institute of Technology has been reported by Shavelson, et al (1990).

Hypermedia—Design Issues

One of the concerns in designing hypermedia systems is navigation (Jonassen, 1988; Marchionini, 1988; Smith, 1988), "knowing where one is, where one wants to go, and how to get there from here," (Parunak, 1989, p. 47). Because of the passive nature of the links provided by such systems, the user has to choose to pursue some path. This concern over whether the student will choose to pursue the appropriate path to learn important material raises a number of issues. First, the student must recognize that he/she needs help in learning something and decide that it is worthwhile to do so. Second, he/she must decide where to go to learn this material.

Thus, giving the user control is a double-edged sword. It may reduce tedium and give the student a sense of control. It does not, however, ensure that important material will be viewed, let alone learned, and "it is not clear how hypermedia can best support learning and instruction" (Jonassen, 1992, p. 4). A great deal of research remains to be done on how to structure hypermedia (Gordon & Gill, in press) and how to influence students to make appropriate use of the available links.

Level-3 Interactive Video Systems

"Hyperscience 456" (Hofwolt, Kumar, & Altman, 1991) is an example of an application of level-3 interactive video technology. Hyperscience presents counterintuitive events or discrepant events (using a video disk) in order to stimulate curiosity, wonderment, critical thinking, and a need for students to seek explanations for the observed phenomena. The video disk interacts with a HyperCard (TM) stack in a Mac II computer via a Pioneer 4200 videoplayer, with images displayed on a Sony color television.

Instead of looking at still photographs, in Hyperscience the learner gets a first hand view of the counterintuitive event in action on a color TV monitor. This capability enriches the context of the learner-machine interaction. The topics include Air and Pressure, Buoyancy, Characteristics of Matter, Heat, Light, Magnetism, Mechanics, Sound, and Earth Science. An example of a Hyperscience 456 stack arrangement is shown in Figure 5.

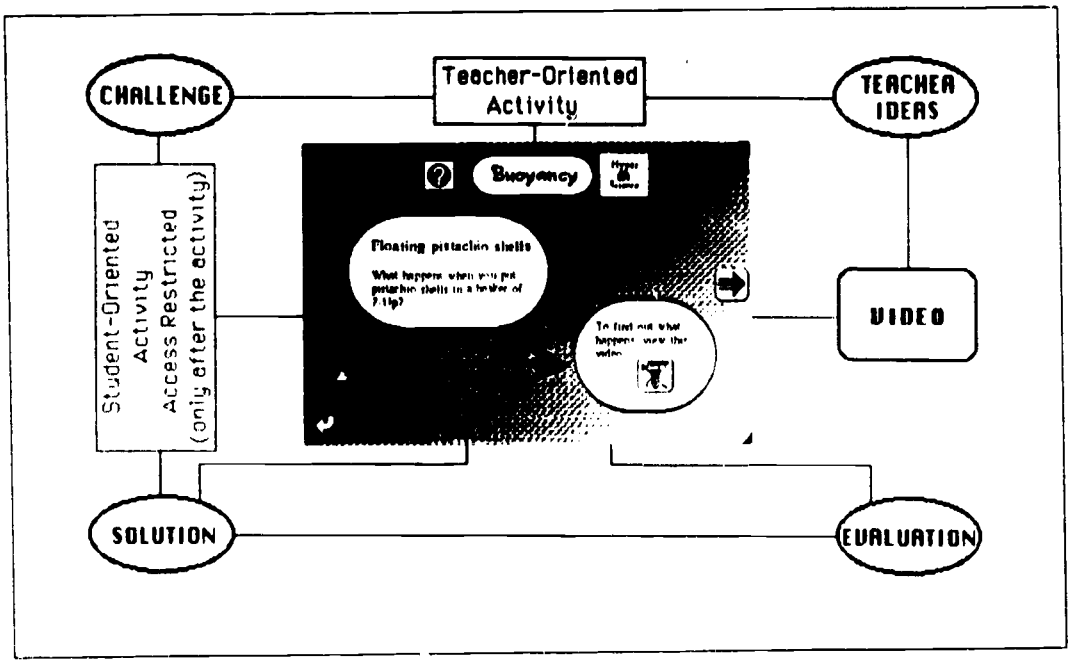


Figure 6. Schematic diagram of Hyperscience *(Hyperscience was first presented at the NSTA convention held in Houston TX, 1991)*

Hyperscience is designed to support the teacher in introducing and reinforcing specific science concepts, and in teaching problem-solving. It does so in several ways. First, a catalog of carefully selected problems has been assembled for use by the teacher. Second, appropriate teaching strategies such as discovery learning and verification experiments using the laboratory are suggested to the teacher. Third, videos of the counterintuitive events make them "real" and serve to stimulate thought and discussions by the students. The normal mode of use is for the teacher to lead a class discussion using Hyperscience to display video segments, or for the teacher to circulate among groups of students (each at a computer workstation) and to engage them in group discussions. Outcome studies are in progress.

This use of the computer to facilitate instruction and to stimulate discussions is further evident in the "Teacher Education Project" (Goldman & Barron, 1990). In the Teacher Education Project, an interactive video environment is employed to present videos of contrasting instructional strategies and to initiate discussion among preservice science teachers at Vanderbilt University. According to Goldman and Barron (1990), preservice teachers who used the interactive videos in their methods class improved considerably in classroom management practices and in several instructional strategies such as development of problem solving skills and higher order cognitive skills.

The use of interactive video technology provides the learner with the opportunity to go back over scenes and review the events. This can be particularly useful in problem solving exercises in which the person is unable to note and remember all of the pertinent information relevant to the solution of the problem. It is also possible to "mark" a certain point on the computer monitor or to make measurements of events shown by the image on the videodisc.

Events which take an inordinately long or short time can be slowed or speeded up for more meaningful and reasonable observation within the time and facility restraints of the classroom. Dangerous and otherwise inaccessible systems can be accessed and manipulated through computer and videodisc technology. This allows students to experience events that would otherwise be beyond their realm of personal experience by providing a concrete, personalized experience to better understand important concepts in science.

Level-3 Interactive Video—Design Issues

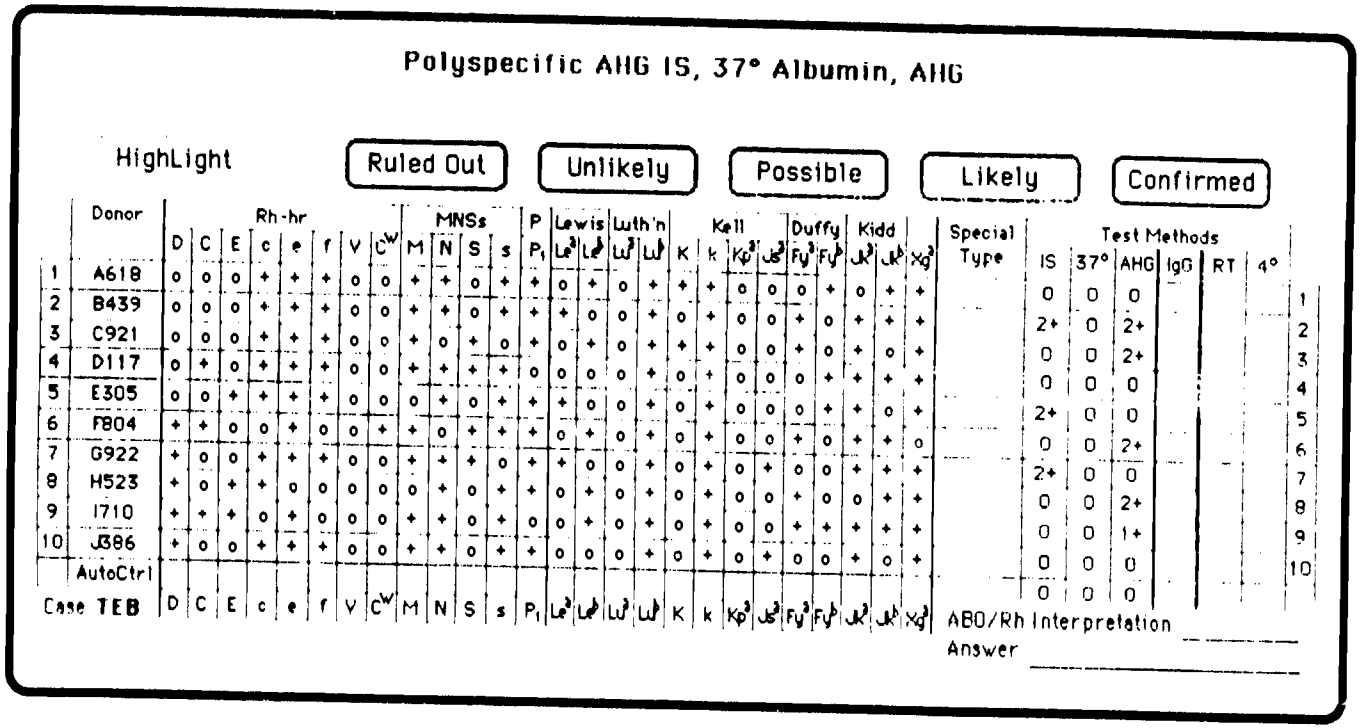
The design of level-3 interactive video raises a number of interesting questions about the use of computers in education:

1. What role should the teacher, the student and the computer play? For example, Hyperscience contrasts with traditional approaches to computer-aided instruction (CAI) in that the teacher is actively involved in the on-going activities, probing, providing feedback, motivating, and directing discussions. Although, like traditional CAI, Hyperscience provides a question and an answer, the teacher's role is enforced because students do not enter an answer on the computer. Rather, they discuss their ideas with the teacher. Thus, teachers can ask questions such as "What do we know about this event?" "What do we need to find out?" and "How are we going to find out?" in order to encourage students to perform their own investigations and arrive at possible explanations. In addition, after developing the relevant background, the teacher may pose questions such as "How can we use this information?" and invite learners to explain their answers and responses to the class for further discussions.
2. Does observing video of an event (as opposed to the text-based description of the event) stimulate and enhance reasoning and problem solving about that event, and also lead to better long-term retention of lessons learned? In other words, does the display medium affect the learning process?
3. How does embedding problem solving in a concrete setting through presenting the problem episode in a video (as opposed to an abstract description of the relevant phenomenon) improve learning?

An Intelligent Tutoring System

To highlight issues associated with the design of ITSs, consider the Transfusion Medicine Tutor (TMT). Written in C and running on a Mac II with three color monitors, TMT provides a problem solving environment similar to GAIT. In the case of TMT, however, the problem solving task is the identification of antibodies in a patient's blood. This is a complex abduction task in which masking and noise combine to pose a challenging problem solving task (Smith, Galdes, Fraser, Miller, Smith, Svirbely, Blazina, Kennedy, Rudmann & Thomas, 1991).

The left screen displays the tests normally available to a technologist in a transfusion laboratory as shown in Figure 6 (Smith, Miller, Fraser, Smith, Svirbely, Rudmann Strohm, & Kennedy, 1991). The center screen displays the particular test result selected for viewing from the set of tests on the left screen. (See Figure 7 for an example.) The right screen is used for selecting a final answer and for tutoring. (See Figure 8).



BEST COPY AVAILABLE

Figure 6. Sample TMT data display

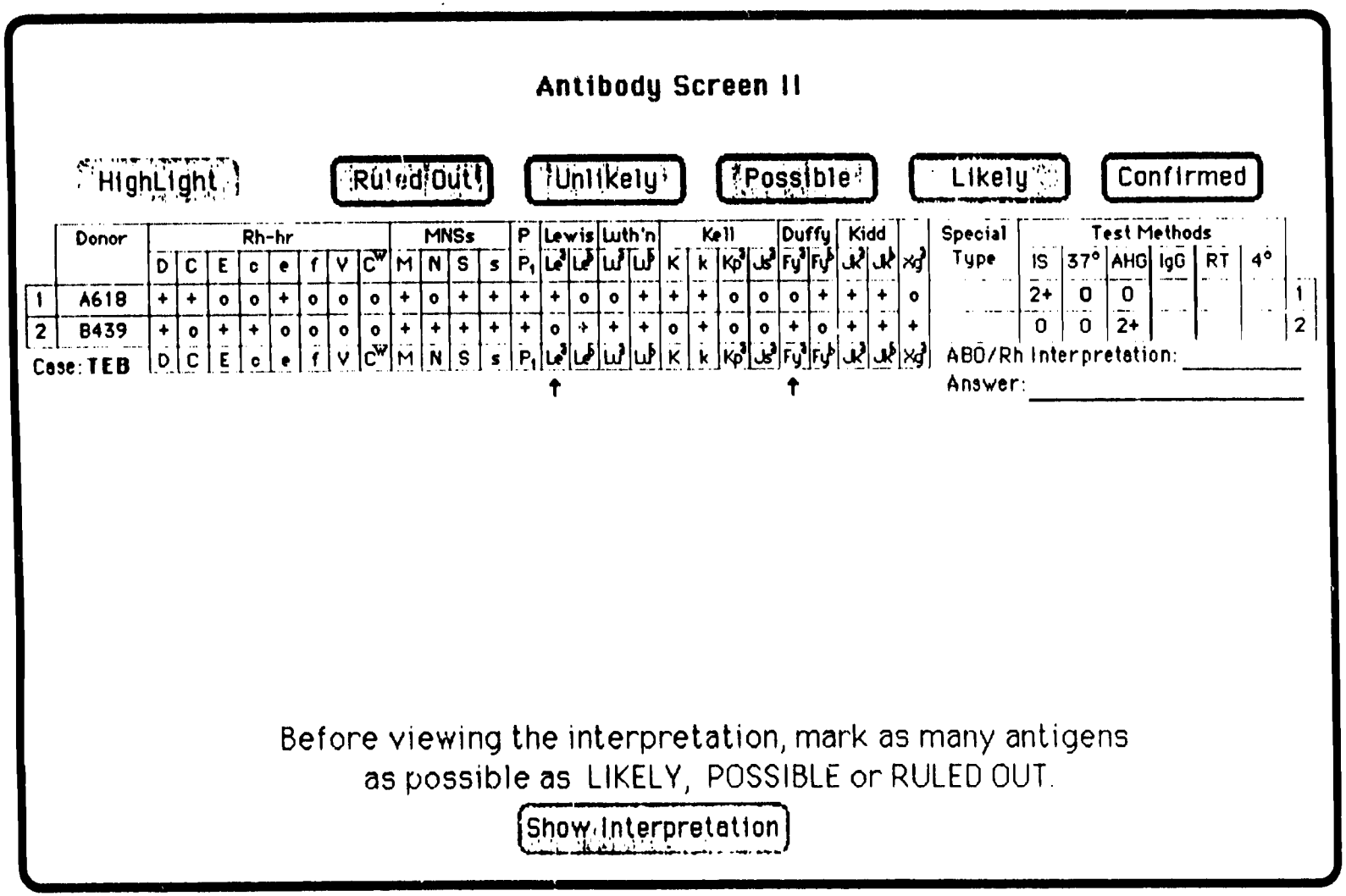


Figure 7. Sample display of the feedback to student

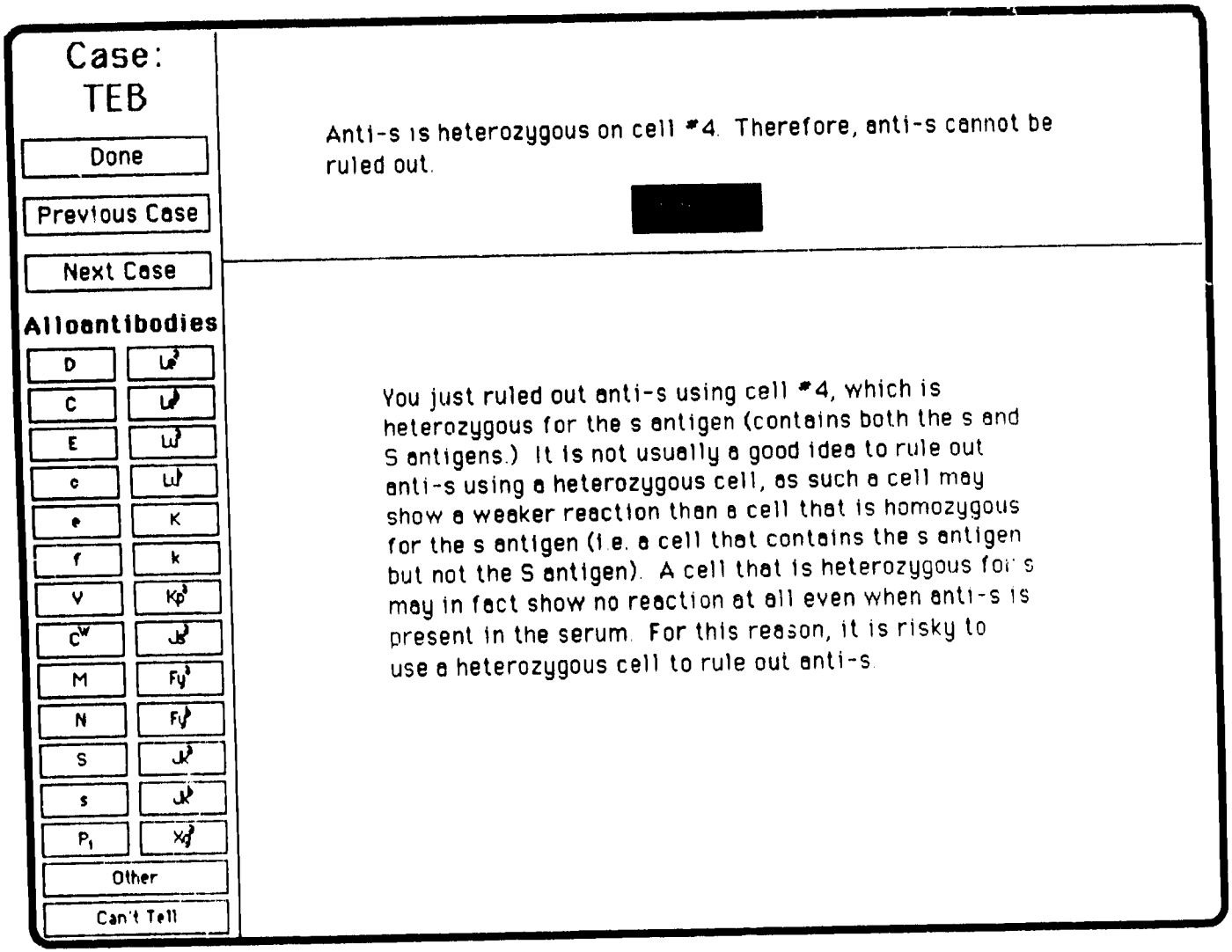


Figure 8. Message in response to an erroneous rule-out

When data such as those shown in Figure 7 are displayed on the center screen, the student can mark intermediate conclusions in a manner analogous to markings presently made on paper in the laboratory. He/she can highlight data, mark antibodies as ruled out, etc. As a memory aid, these intermediate conclusions are propagated from one test to another as the student explores various test results.

TMT differs from GAIT in that it has an expert system embedded in its architecture. This expert system monitors the student markings (intermediate and final conclusions) and provides feedback in response to errors. It also provides a discussion of an expert's interpretation of any given set of data upon request.

Due to recent concerns related to the transmission of blood related diseases it is no longer reasonable to conduct blood typing activities in biology classrooms using the students' own blood. Use of this technology as in TMT enables students to explore blood typing in a safe environment.

Tutoring Function

TMT monitors student inferences and requests for data. These actions are used to detect errors (actions that run contrary to those of the expert model). Examples of such tutoring behaviors are categorized below.

Inappropriate Test Selection. One class of actions performed by the student is a request to run a particular test. TMT uses the data currently available to the student about that case, plus its knowledge about the appropriate use of tests, to assess such decisions. If the student has requested an inappropriate test, TMT detects the error and interrupts the student. The interruption consists of a caution and an explanation of the basis for this caution. TMT also provides suggestions about what to do next.

Testing for Understanding. Preliminary studies of student performances indicated that students sometimes know enough to ask for the right test, but not enough to fully interpret the results. Our expert human tutors frequently detected this by asking a question at the appropriate point. TMT does likewise, presenting multiple choice questions at points where students are likely to have misunderstandings.

Erroneous Intermediate Conclusions. TMT also monitors for errors of omission and commission. Since the student can mark intermediate conclusions on the displayed data sheets, TMT can check to see whether the appropriate conclusions have been drawn. TMT monitors for two types of errors: Drawing an incorrect intermediate conclusion (such as incorrectly ruling out an antibody), and failing to draw a conclusion that the data support.

Erroneous or Questionable Final Conclusions. TMT also looks at the student's final answer (indicated by clicking on buttons representing possible antibodies) and critiques it. If, for example, the student has concluded anti-C is present alone in a case where anti-C and anti-D are present, the system will point out the error. As part of this critiquing process, TMT teaches the student methods for detecting his/her own errors.

TMT—Design Issues

The design of TMT raises a number of important questions. First, like the design of Hyperscience, the role of the teacher must be defined. Informal evaluations of the TMT suggest that its most effective use is not as a stand-alone teaching system, but as a learning environment in a laboratory setting, where the teacher can circulate among students working on TMT, asking questions and providing assistance.

A second issue involves how to design a system that can detect students' errors in a timely fashion. The interface to TMT was explicitly designed to enable such error detection. Because students have to request specific pieces of data and draw intermediate conclusions, TMT can detect many errors immediately without being intrusive. Other types of naive conceptions are handled by having the system actively probe with a question. A third issue is the question of when to interrupt given that an error has been detected. Empirical studies of expert human tutors suggest that such decisions involve complex reasoning (Galdes, Smith & Smith, 1990) which is beyond TMT's current capabilities.

A fourth issue is raised by the use of the colored arrows used as feedback by TMT (in Figure 7). The philosophy behind this design feature is that students should be given the opportunity to develop their own explanations before looking at the computer's (Chi, Bassok, Lewis, Reiman, & Glaser, 1989).

A fifth issue concerns the adoption of alternative learning strategies by students. Will they simply look at the computer's answer or will they try to interpret the data on their own first? What will they read of the computer's explanations? How do we influence them to adopt effective learning strategies and help them modify their own learning strategies to be effective? The literature on how people use documentation suggest that these are non-trivial concerns (Wright, 1983). Multi-media feedback may be part of the answer.

Future Directions

The advanced technologies discussed above offer two approaches to enhance learning. Midro, Chiacariello, Olimpo, Perisco, Sarti, and Tavella (1988) suggest that the integration of these technologies can alleviate many of the shortcomings of hypermedia (including level-3 interactive video) and intelligent tutors, and lead to the development of more intelligent and flexible systems capable of making teaching and learning more efficient and meaningful.

Currently emerging technologies add yet further promise of increased flexibility and efficiency. One example is the pen-based computer. These computers are completely wireless, responsive to handwritten input, and capable of interaction with other systems. The applications of this technology with its wireless portability and reduced interface barriers are just beginning to be explored. It seems clear, however, that the instructional potential of the pen-based computer is high, particularly when combining it with other approaches such as hypermedia and intelligent tutoring systems.

Conclusions

The use of hypermedia (including level-3 interactive video), and the integration of "intelligence" into tutoring systems, represent two important approaches for enhancing the use of computers as educational tools. To date, however, most of the effort has gone into exploring the implementation of such systems in attempts to identify alternative capabilities and uses of these technologies. The result has been a number of interesting models. Informal evaluations indicate that these approaches can offer significant improvements over traditional uses of computers in education. As outlined in this paper, however, there are numerous questions which remain to be answered in order to develop an empirical basis for guiding the design of such learning environments. For example, Clarke (1990) in a

review of computer usage found sex discrimination favoring male students. To what extent this discrimination is revealed with these advanced systems remains to be determined.

GAIT, Hyperequation Project, and similar on-going projects are relatively novel applications of hypermedia in developing alternative science assessment technologies. Hyperscience 456 and Teacher Education Project may be classified as examples of learning with computers (Luehrman, 1982 February, September), and one of the most useful applications of "level 3IVD" systems. The Transfusion Medicine Tutor, an example of building intelligent tutors using expert systems technology, is a very promising practical application for science education.

Such systems are nothing more than advanced technologies. Therefore, how they are used in science instruction will determine their future success in education. The applications presented in this paper are a glimpse of what these advanced technologies can do for education. These technologies offer great hope for science education of tomorrow, and offer the potential to transform science learning into a meaningful, interesting, and practically relevant experience. To accomplish this goal, however, we must go beyond the implementation of interesting systems. We need to use such systems as testbeds to collect empirical data on the effectiveness of the underlying design concepts.

References

Aambo, K. H., & Hovig, I. (1988). The term hypermedia & thought-experiment hypatia. *Proceedings of Research in Networks and Distributed Applications* (pp. 259-270). Wien, Amsterdam: North-Holland.

Ambron, S., & Hooper, K. (1988). *Interactive multimedia*. Redmond, WA: Microsoft Press.

Anderson, J., Boyle, C., & Reiser, B. (1985). Intelligent tutoring systems. *Science, 228*, 456-462.

Barrows, H. S. (1988). A taxonomy of problem based learning. *Medical Education, 20*, 481-486.

Boring, J. R., & Nutter, D. O. (1984). Analytical thinking: Educating students for the practice of modern medicine. *Journal of Medical Education, 59*, 875-880.

Burnett, C. N., Mahoney, P. J., Chidley, M. J., & Pierson, F. M. (1986). Problem solving approach to clinical education. *Physical Therapy, 66*, 1730-1733.

Bush, V. (1945, July). As we may think. *Atlantic*, pp. 101-108.

Cardiff, R. D. (1986). Teaching problem solving in pathology. *Archives of Pathology and Laboratory Medicine, 110*, 780-783.

Chi, M., Bassok, M., Lewis, M., Reiman, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13*, 145-182.

Clarke, V. A. (1990). Sex differences in computing participation: Concerns, extent, reasons and strategies. *Australian Journal of Education, 34*(1), 52-66.

Clancey, W. (1984). Methodology for building an intelligent tutoring system. In W. Kintsch, G. Miller, & P. Polson (Eds.), *Methods and tactics in cognitive science*. Hillsdale, NJ: Erlbaum.

Collins, A., Warnock, E., & Passafiume, J. (1975). Analysis and synthesis of tutorial dialogues. In G. Bowe (Ed.), *The psychology of learning and motivation*. NY: Academic Press.

Conklin, J. (1987). Hypertext: An introduction and survey. *IEEE Computer, 20*(9), 17-41.

Dede, C. J. (1987). Empowering environments, hypermedia and microworlds. *The Computing Teacher, 15*(3), 20-24.

Galdes, D., Smith, P. J., & Smith, J. W. (1990). Building an intelligent tutoring system: Some guidelines from a study of human tutors. *Proceedings of the 1990 Annual Conference of the Human Factors Society*, pp. 1407-1411.

Gilman, D. A., & Brantley, T. (1988). *The effect of computer assisted instruction on achievement, problem-solving skills, computer skills, and attitude.* Terre Haute, IN: Indiana State University. (ED 302 232)

Glusko, R. (1989). Design issues for multi-document hypertexts. *Proceedings of Hypertext '89* (pp. 51-60). NY: ACM Press.

Goldman, E., & Barron, L. (1990). Using hypermedia to improve preparation of elementary teachers. *Journal of Teacher Education, 41*(3), 21-31.

Gordon, S., & Gill R. (in press). Knowledge acquisition with question probes and conceptual graph structures. In T. Gauer, E. Peacock, & A. Graesser (Eds.), *Questions and information systems.* Hillsdale, NJ: Erlbaum.

Halasz, F. G. (1988). Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM, 31*(7), 836-852.

Halasz, F.G., Moran, T.P., & Trigg, R.H. (1987). Notecards in a nutshell. *Proceedings of the ACM/SIGCHI conference on Human Factors in Computing Systems* (pp. 45-52). New York: Association for Computing Machinery.

Henry, J. N. (1985). Identifying problems in clinical problem solving. *Physical Therapy, 65,* 1071-1074.

Hofwolt, C. A., Kumar, D. D., & Altman, J. E. (1991). *Hyperscience 456.* Paper presented at the annual meeting of the National Science Teachers Association, Houston, TX, March 1991.

Irby, D. M. (1986). Clinical teaching and the clinical teacher. *Journal of Medical Education, 61,* 35-45.

Johnson, W. L., & Soloway, E. (1985). PROUST: Knowledge-based program understanding. *IEEE Transactions on Software Engineering, 11,* 267-275.

Jonassen, D.H. (1986). Hypertext principles for text and courseware design. *Educational Psychologist, 21*(4), 269-292.

Jonassen, D.H. (1988). Designing structured hypertext and structuring access to hypertext. *Educational Technology, 28*(11), 13-16.

Jonassen, D. H. (1992). Learning vs. information. *Journal of Educational Multimedia and Hypermedia, 1*(1), 305.

Kearsley, G. (1987). *Artificial intelligence and instruction.* Reading, MA: Addison-Wesley.

Kumar, D. D. (1991a). Hypermedia: A tool for STS education? *Bulletin of Science, Technology & Society, 11,* 331-332.

Kumar, D. D. (1991b). *A note on "Hyperequation."* Unpublished manuscript. National Center for Science Teaching and Learning, Columbus, OH.

Litchfield, B. C., & Dempsey, J. V. (1992). The IVD-equipped classroom: Integrating videodisc technology into the curricula. *Journal of Educational Multimedia and Hypermedia, 1*(1), 39-49.

Luerhman, A. (1982, February). *Microcomputers and children.* A paper presented at the Wingspread Conference on Microcomputers in Education, Wingspread Conference Center, WI.

Luerhman, A. (1982, September). Don't feel bad about teaching BASIC. *Electronic Learning,* pp. 23-24.

Marchionini, G. (1988). Hypermedia and learning: Freedom and chaos. *Educational Technology, 28*(11), 8-12.

Marsh, E. J., & Kumar, D. D. (1992). Hypermedia: A conceptual framework for science education and review of recent findings. *Journal of Educational Multimedia and Hypermedia, 1*(1), 25-37.

Martinez, M. E. (1991). Figural response in science and technology testing. In G. Kulm, & S.M. Malcom (Eds.), *Science assessment in the service of reform*. Washington, DC: AAAS.

Midro, V., Chioccariello, A., Olimpo, G., Perisco, D., Sarti, L., & Tavella, M. (1988). Interactive video and artificial intelligence: A convenient marriage. *Programmed Learning and Educational Technology, 25*(4), 299-309.

Moore, J. W. (1989). The FIPSE lectures: Tooling up for the 21st century. *Journal of Chemical Education, 66*(1), 15-19.

Mulhauser, M. (1992). Hypermedia and navigation as a basis for authoring/learning environments. *Journal of Educational Multimedia and Hypermedia, 1*(1), 51-64.

Norman, D. (1988). *The psychology of everyday things*. New York: Basic Books.

Parunak, H. (1989). Hypermedia topologies and user navigation. *Proceedings of Hypertext '89* (pp. 43-50). NY: ACM Press.

Shavelson, R. J., Baxter, G. P., Pine, J., Yure, J., Goldman, S. R., & Smith, B. (1990). *Performance indicators for large-scale science assessment*. Paper presented at the annual meeting of the American Educational research Association (Session #37.15), Boston, April 1990.

Sleeman, D., & Brown, J. (Eds.). (1982). *Intelligent tutoring systems*. New York: Academic Press.

Smith, K.E. (1988). Hypertext—Linking to the future. *Online, 12*(2), 32-40.

Smith, P. J., Miller, T. E., Fraser, J., Smith, J. W., Svirbely, J. R., Rudman, S., Strohm, P. L., & Kennedy, M. (1991). An empirical evaluation of the performance of antibody identification tasks. *Transfusion, 31*, 313-317.

Smith, P. J., Galdes, D., Fraser, J., Miller, T., Smith, J. W., Svirbely, J., Blazina, J.. Kennedy, M., Rudmann, S., & Thomas, D. L. (1991). Coping with the complexities of multiple solution problems: A case study. *International Journal of Man-Machine Studies, 35*, 429-453.

Tsai, C. (1988). Hypertext: Technology, applications, and research issues. *Journal of Educational Technology Systems, 17*(1), 3-14.

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational approaches to the communication of knowledge*. Los Altos: Kaufman.

Woolf, B. (1987). Theoretical frontiers in building a machine tutor. In G. Kearsley (Ed.), *Artificial intelligence and instruction*. Reading, MA: Addison-Wesley.

Woolf, B., & McDonald. (1985). Building a computer tutor: Design issues. *AEDS Monitor, 23*, 10-18.

Wright, P. (1983). Manual dexterity: A user-oriented approach to creating computer documentation. *CHI'1983 Conference Proceedings*, Boston, 11-18.

Acknowledgements: Preparation of this manuscript was supported by the National Center for Science Teaching and Learning under OERI Grant No. R117Q00062, U.S. Department of Education. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsoring agency. Appreciation is also due to Susan Gross, Stephanie Guerlain, Tom Miller, Salley Ruddmann, Pat Strohm, John Svirbely, Jack Smith, Shelly Simon, Jean Nippa, Melinda Green and Dan Rosenberg for their work in developing TMT and GAIT, to Clifford Hofwolt and Jan Altman for their collaboration in developing Hyperscience, and to John Harwood for his assistance in developing Hyperequation.

Chapter 4

Videodisc Technology: Applications for Science Teaching

DERRICK R. LAVOIE

"One in a hundred thinks, one in a thousand sees."

Joseph Albers

An ancient milestone transpired when a primitive humanoid first used a piece of charcoal from the fire to represent knowledge as crude visualizations on the walls of a cave. Today, new technologies are defining another milestone in the development of visual learning.

This chapter establishes a theoretical framework for visual learning and the need for visual information processing in science based on a cognitive-science/constructivistic perspective. The discussion focuses on practical "how to" techniques for science teachers interested in designing, developing, and applying exciting visually-rich learning experiences in their classrooms via videodisc technology. Emphasis is placed on the development of conceptual understanding, science-process skills, and problem-solving skills. The final part of the chapter poses several questions for the future of videodisc technology in the science classroom.

A Constructivistic/Information-Processing Theoretical Framework

How do my students learn? This is the most fundamental question to be addressed by science educators concerned with improving their instruction. The relatively new field of cognitive science attempts to answer this question from an information-processing/

constructivistic perspective. Learning is viewed as an active process in which the students, themselves, must construct "meaningful" understandings (Ausubel, 1963; Shuell, 1990). In this process, knowledge is selected, discriminated, associated, and elaborated within a previously existing knowledge structure or cognitive network.

Generally, cognitive scientists believe this cognitive network to be composed of both declarative and procedural knowledge (Anderson et al., 1990; Derry, 1990; Mayer, 1989). Declarative knowledge, or product knowledge, is associated with facts and concepts. Procedural knowledge, or process knowledge, is associated with knowing how to do something.

Ultimately, it is the nature of the relationships between students' procedural and declarative knowledge that determines their success relative to an achievement test, designing an interesting science experiment, or solving a problem. In successful cognitive networks, procedural knowledge is linked to, acts on, and applies declarative knowledge in productive ways. Thus, meaningful learning and, it follows, meaningful teaching must be a continual process of constructing and reconstructing successful relationships between relevant declarative and procedural knowledge. Visualization is a powerful tool that can facilitate this process.

The Importance of Visualization

A plethora of research points to the advantages of visual aids in developing scientific understanding (Brody, 1984; Dwyer, 1972; Holliday, 1975; Rigney & Lutz, 1976), memory encoding (Houston, 1991; Vasu and Howe, 1989), and motivation (Kozma, 1991). An important finding of this research indicates it is not enough to simply show students a picture or movie—there must be a *purposeful interactivity,* or relevant information-processing demand, between the learner and the visual. The goal of this interactivity is to draw out important ideas, stimulate creative thought, and promote meaningful conceptual linkages within the provided context.

Relative to cognitive science, purposeful visual processing must somehow lead to more elaborated and explicit knowledge relationships than are possible through audio, verbal, or textual means. Perhaps, due to the inherent high density of information contained in visuals, there is more opportunity for establishing linkages with pre-existing knowledge. Such linkages may be iconic to verbal, iconic to iconic, etc. The high information density could also account for research findings indicating that while visual processing takes more time than textual (Shapiro, 1985) it is a more efficient way to learn (Paivio, 1971).

Einstein noted that at the highest level of his thinking was pictorial symbols. Possibly, the architecture of the brain can store iconic information more efficiently than verbal information, as well as facilitate the linking of such information with other knowledge. This might, in part, explain the positive learning results obtained for concept mapping (Fisher, 1990; Mason, 1992; Novak & Musonda, 1991;). Concept maps can be considered iconic representations of cognitive schemata. It also seems likely that visual information is easier to "chunk" and manipulate in short-term memory. Other findings report a "parallel processing" advantage of visual over other learning modes (Reed, 1992).

In sum, the research suggests that the teaching of science should be done concurrently with visual aids during all phases of instruction and at all levels of cognitive development. Videodisc technology, combined with appropriate teaching/learning strategies, can be used to achieve high levels of visual interactivity.

Videodisc Technology

A typical videodisc is composed of a variety of simulated/animated processes, still pictures, motion pictures, and graphs that can be accessed almost immediately from anywhere on the disk via a computer or remote control. Animated or real-life sequences can be slowed down, sped up, frozen, manipulated, and experimented with in ways uniquely different from videotape or printed media. Videodiscs offer microworlds within which students can explore and discover concepts in "real world" contexts—contexts that would otherwise be unavailable to science students due to concern for safety, time, expense, or accessibility.

Most commercial 12 inch laser videodiscs in the sciences cost from $100 to $600 and contain 54,000 numbered picture frames per side. While these visual databases may involve 54,000 discrete slides, only 30 minutes of motion picture is possible on a one-sided videodisc. Distributors of videodiscs for science include Videodiscovery (206-285-5400) and Optical Data Corporation (1-800-524-2481).

Learning and Teaching via Videodisc Technology

Videodisc presentation format is classified based on the degree of remote control (Weller, 1988). At the most basic level of presentation, the videodisc is basically played and stopped, and played again, much like videotape. This form of instruction is very linear and visually deductive, leaving little room for student directed learning. At a higher level a teacher or student using an access controller can immediately search and explore information anywhere on the videodisc. They can illustrate particular concepts, provide interesting presentations, and engage in visually-rich problem-solving experiences. This allows for any videodisc segment or frame to be accessed almost immediately. The highest level of user control commonly employs a videodisc player connected to a microcomputer that is programmed to interact with the user, based on his/her actions, and adapt instruction accordingly.

Teachers or computers can increase visual interactivity by asking appropriate divergent and convergent questions, giving opportune feedback, posing visually-oriented problems, and using a visually-rich medium such as videodisc technology. It will be useful for teachers to consider visual interactivity based on a hierarchy of increasing information-processing demand:

Level 1- Motivation, interest, appeal.
Level 2- Explanation or exploration of a concept, relationship, or procedure.
Level 3- Generation of questions, hypotheses, predictions or inferences.
Level 4- Experimentation, data manipulation and collection
Level 5- Problem solving, evaluation, metacognition

As with Bloom's taxonomy, teachers will want to engage students at a variety of levels for optimal success. As with other inclusive hierarchies of cognition, the highest level of interactivity involves problem solving. In this case, solving the problem requires interaction with a visual that would entail processing from Level 1 to Level 4 above.

Teachers should also find it useful to consider learning, via videodisc, on a continuum from visual deduction to visual induction. Visual deduction requires students to identify

a predetermined concept while viewing still or motion videodisc segments, also predetermined by the teacher. Visual induction, the more student-structured strategy, involves students exploring and piecing together images from the videodisc to represent their *own* understanding of a particular science concept which may or may not have been selected by the teacher. This is similar to the "imposed picture strategy" of Alesandrini and Rigney (1981) and representative of "constructivistic" learning strategies (Basili, 1989; Wheatly, 1991; Glynn, Yeany, & Britton, 1991). The student-structured nature of visual induction tends to access higher levels of the visual information-processing hierarchy. Thus, it is seems likely that visually inductive strategies are more effective for revealing students' prior knowledge (pre-scientific conceptions) and generating conceptual understanding than are visually deductive strategies.

Another distinction between visual deduction and induction concerns linearity. While the learning pathway of visual deduction tends to be more "linear," learning via visual induction can be much more "non-linear." The non- linear learning experience is fundamentally different from the traditionally linear mode in that the teacher or the computer does not know what the student will do (Nix, 1990). Relative to the need for non-linearity, Spiro and Jehng (1990) comment:

> The advent of random access computer technologies makes practicable new forms of nonlinear and multidimensional learning and instruction that are better suited to conveying complex content. (p. 163)

Non-linear instruction seems to be a key for going beyond what is explicitly taught by requiring learners to independently apply their knowledge in a pattern that is not preset by text or teacher. The flexibility of the videodisc learning environment can provide a variety of visual arenas within which students can choose a infinite number of learning pathways.

Videodisc instruction, without computer control, can become an effective vehicle for delivering "anchored instruction" (Bransford et al., 1990). Anchored instruction is a kind of visually deductive strategy that provides the student with a motivating focus requiring the identification and application of visual information that is relevant to solving a posed problem. An effective anchor forces the student to notice more relevant features or patterns of a problem, to develop more complex search paths, to identify relationships between more information, and to become metacognitively conscious of changes in his/her own cognitive processing and understanding. Information-processing models consider this to result in conditionalized knowledge—knowledge stored in the form of production systems linked to goal-oriented problem-solving processing (Anderson, 1987; Holland et al., 1985; Simon, 1980). Productions systems consist of condition-action (if-then) pairs that link declarative (factual knowledge) and procedural knowledge (process knowledge). When students conditionalize their knowledge it is viewed as a means to an end and not an end in itself.

In the case of visual interactivity, procedural knowledge acts on and establishes links with visual declarative knowledge which may then be further linked to verbal or audio declarative knowledge. Some research indicates visual-based learning strategies, while improving visual processing, may not necessarily improve verbal processing (Alesandrini & Rigney, 1981). Blystone and Dettling (1990) list several good

suggestions for critiquing and using visuals with science textbooks which have direct implications for visual-to-verbal and verbal-to-visual processing.

Lastly, science teachers using videodiscs for instruction should find it useful to consider the literature concerned with improving students' visual-spatial skills (Hassard, 1982; Lord, 1987; Lowery & Knirk, 1982;), techniques for creative visualization (Finke, 1990; Zielinski & Sarachine, 1990), and visual-learning modes (Vasu and Howe, 1989).

The next section of this chapter describes several inductive and deductive strategies science teachers can use with videodisc anchors to achieve high visual interactivity with and without computer control. Economical techniques are also described for producing your own videodiscs.

Visually Deductive Strategies

Research efforts by the Cognition and Technology group at Vanderbilt University have resulted in several videodisc anchors for science instruction which take the form of problem-solving vignettes (Bransford et al., 1990; Sherwood, 1991). For example, using a videodisc of "Raiders of the the Lost Ark," students are asked to consider problems Indiana Jones might encounter on a trip through the South American jungle. Another videodisc, which was actually produced by the Vanderbilt group, follows a family on a one-week trip down a river. Students identify and solve a series of sub-problems relative to determining the required food, water, gas, and destination distance. It's an integrated approach requiring knowledge of math and science. At least fifteen different bits of embedded knowledge must be identified on the videodisc.

Recently, I developed several problem-solving vignettes on videotape with the help of high-school science students. The following production sequence, involving students working cooperatively on each phase of the vignette production, considerably adds to their learning, motivation, and excitement about science.

First, students identify an interesting problem situation in science. Second, they write a script to include an interesting story line involving a small group of "acting" investigators that encounter the problem situation. One high-school biology class I worked with developed a vignette called "Bad Water," which involves a group of high-school students taking action to determine the effect of a toxic leak in their local water source. At the end, a problem is posed challenging the viewers to decide how they would deal with the problem. Information relevant to solving sub-problems associated with stopping the leak, determining the extent of the contamination, and dealing with the contamination is embedded in the videotape. Science concepts center around diffusion, waste management, ground water contamination, and proper sampling methods.

Third, a cast of actors and actresses, the stage crew, and the video production crew (all made up from your students) go on location (e.g., a stock room or school creek) to record the drama of each scene on videotape. Students always find this part most entertaining. The stage crew must be sure all materials and equipment are set up for each scene. The video production crew maintains proper lighting, background, and audio levels while directing the videotaping. Of course, the cast must have rehearsed their parts. The videotaped scenes usually must be edited to a final videotaped version which can be used for anchored instruction as described above. The tape can offer more flexibility and accessibility by being transferred to a videodisc.

Visually Inductive Strategies

A project I am currently working on involves producing a videodisc from videotaped episodes of the elementary classrooms to be used in a new course at Montana State University. The goal of the course, in the spirit of "reflectionism," constructivism, and visual induction, will be to give Freshmen entering the teacher education program an early experience formulating initial ideas and perspectives about teaching (i.e., developing a prior experiential knowledge base). The students will use the videodisc anchor to create computer-controlled Hypercard presentations illustrating key concepts which *they* identify from the videodisc. Hypercard can become an effective vehicle for these kinds of visually inductive learning experiences. Using its user-friendly iconic environment, it is relatively easy for students to select and sequence videodisc segments, add text or graphics, and piece together interesting presentations that reveal their understandings.

Videodisc Production Strategies

While videodiscs may be purchased commercially, the innovative science teacher working from a modest budget, can actually create his/her own videodisc for under $500. You will need two 1/2" VCRs, a high-quality 1/2" VHS camcorder (super VHS is best), and a 3/4" editing machine. After shooting your videotape footage, it is likely you will need to edit the appropriate vignette sequence to another tape while removing any excess or confusing footage. Video and audio RCA connecting cables should be plugged into the "out" ports on the "play" VCR and into the "in" ports on the the "record" VCR. The edited vignette is made by recording appropriate video sequences on the "record" VCR from the original videotape that is played on the "play" VCR. Although you will lose one "generation" of quality, the final video vignette is more than adequate for use in the classroom. Next you must dub the 1/2" VHS tape up to a 3/4" tape while adding color bars and time codes. In the event you do not have direct access to a 3/4" editing machine, most video production companies will do it for about $150. The final 3/4" 30 minute videotape is sent to a production company such as 3-M Optical Recording or Optimus, Inc. to be formatted to a videodisc for about $400. Most videodisc formatting companies will send specifications for the final videotape, prices, and other important information.

Interactive Videodisc Lessons for Higher Level Visual Interactivity

A videodisc linked to a computer is commonly referred to as an interactive videodisc (IV). In general, the more the system is adaptive, reactive, and flexible the greater its interactivity. In this case, interactivity can be defined as the exchange of information between the user and the computer.

IV lessons can effectively compensate for lack of teachers and materials, free the teacher for more individualized instruction, and free the students for more hands-on activities. Research indicates IV can substantially improve students' conceptual understanding, problem solving, and psychomotor skills (Smith & Lehman, 1988). Farragher (1991) comments on the possibilities for IV:

> The applications of interactive video to science teaching are unlimited... Individualized tutoring is another obvious application of interactive video. A whole collection of tutorials across all topics in the sciences is envisioned. (p. 13)

The design of this semi-intelligent tutoring lessons should be approached on primary, secondary, and tertiary levels. Primary-level design identifies the basic strategy for effective science teaching science, per se. Secondary-level design is concerned with general principles of instructional design. And, tertiary-level design comprises techniques and strategies that most productively integrate primary and secondary-level design in a computer-controlled environment to create exciting and adaptable interactive lessons.

Primary Level Design

Many national reports have indicated students lack the basic skills to think, reason, and formulate science concepts (Bybee et al., 1989; Mullis & Jenkins, 1988; National Commission for Excellence in Education, 1983; Weiss, 1989). Developing IV lessons to teach at high levels of visual interactivity can help to alleviate this situation. It has been well documented that instruction based on the three-phase Karplus learning cycle (involving phases of exploration, term introduction, and concept application) improves conceptual understanding and process-skill achievement compared to traditional instructional methods (Abraham & Renner, 1986; Lawson, Abraham, & Renner, 1989). Concluding from the research associated with the learning cycle, Lawson, Abraham, & Renner (1989) comment, "It is not only a good way to teach science, it is *the* way to teach science" (p. 77). The learning cycle is thus a good choice for modeling IV lessons.

The Phases of the Learning Cycle

Phase one of the learning cycle, or exploration, encourages students to interact with new concrete situations (e.g., variables) with minimal guidance or demand from the teacher. The situation should raise questions or mental complexities that the students can not resolve which subsequently motivates them toward the discovery of relationships or patterns in data (i.e, the concept). According to Piaget, the mental restructuring that results from progressing from a state of disequilibrium to that of equilibrium is considered to be "true" learning.

Phase two, or term introduction, involves introducing the terms to the students via student discussion, text reading, teacher lecture, etc. This should allow for comparison and clarification of initial experiences with the terms. The patterns and relationships students identified, or at least were exposed to during exploration, are further revealed. Ideally, the teacher should not explicitly state the concept or relationship to be learned. Rather, the students should be led to the concept by careful analysis of the data or logical scientific thinking. A concept, as it should be taught using the learning cycle, can be defined as a mental structure or relationship and its associated term (i.e., verbal label).

In phase three, concept application, students apply their newly acquired concept to additional situations and problem-solving contexts. In essence, by increasing the range of applicability of the concept, the associated cognitive structure is elaborated and strengthened. This reduces the accumulation of unused or "inert knowledge" (Whitehead, 1929). According to current cognitive science views, this process not only increases understanding of the concept, but makes it more accessible in problem-solving situations (Bransford et al.,

1986). Not surprisingly, one of the characteristics of a good problem solver is the ability to identify and apply relevant information.

Recent research by Lavoie (1991) and Lavoie and Good (1988) has improved the effectiveness of the already good learning cycle strategy by adding explicit hypothetico-predictive and discussion phases (see Figure 1). Hypothetico-predictive processing involves making a prediction and supporting it with a reason (i.e., a hypothetical justification). Compared to the traditional learning cycle instruction, this kind of modified learning-cycle instruction results in greater student motivation to carry out scientific investigations, more positive attitudes toward science, more positive attitudes toward their peers, and greater inter-peer interaction during the subsequent phases of the learning cycle. It also leads to significant increases in conceptual understanding, logical thinking abilities, and process skill achievement.

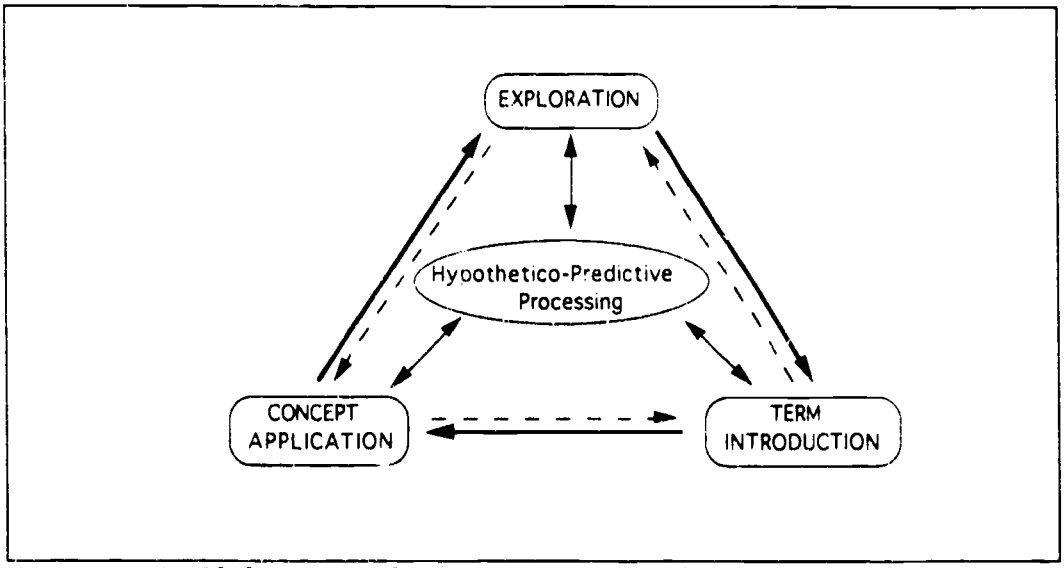


Figure 1. Flexible learning cycle with "hypothetico-predictive" power

Connecting Interactive Videodisc and the Four-phase Learning Cycle

Considering the many advantages of a four-phase learning cycle—including a hypothetico-predictive, exploration, term introduction, and concept application sequence—it is an excellent choice for primary-level design of IV lessons.

For example, the learning cycle might involve an initial prediction phase in which the computer is programmed to display videodisc pictures of scientific events, experiments, or sequences of events. Students would then be prompted to make predictions about the outcomes and to justify their predictions with explanatory hypotheses.

During the exploration phase, IV offers an excellent opportunity stimulate disequilibrium in students as they make detailed and controlled observations of videodisc events. Disequlibrium is considered to be a central component for concept learning by Piaget as well as contempory cognitive scientists. This phase may involve students visually manipulating several variables, in both qualitative and quantitative ways, to observe the real-life effects of their actions. Hassard (1982) offers some simple suggestions to science

teachers for "opening the mind's eye to science" with visualization experiences that are applicable to IV learning.

In term introduction, students may receive both audio and visual definitions of the terms, perhaps while observing a videodisc film or slide sequence illustrating the concepts. During the concept application phase, students could be challenged to apply their conceptual understanding in various videodisc contexts. This may involve finding additional videodisc examples illustrating the concept or answering higher-level questions posed about selected videodisc segments. For example, students might find selected videodisc excerpts that relate the concept to technology and/or societal issues. Also, students could work on a problems requiring them to change parameters and observe the videodisc effects.

Secondary Level Design

In addition to employing the prediction-based learning cycle for the primary-level design, several important instructional design principles based on cognitive learning theory (Gagne et al., 1981; Hannafin & Peck, 1988; Piaget, 1975; Rosenshine & Stevens, 1986), should be considered at the secondary level. These principles imply that computer-assisted instruction should be designed to:

- Gain the attention of the learner.
- Inform the learner of the expectations for learning.
- Stimulate students' recall of prior knowledge.
- Guide the students' learning.
- Present clear and detailed explanations.
- Present lessons at or slightly above students' level of cognitive development.
- Provide the student with systematic feedback that is informative.
- Frequently ask questions at varying levels of Bloom's taxonomy.
- Provide ample opportunity for students to practice.
- Assess and monitor student performance.
- Maintain congruence between objectives, instruction, and assessment.
- Address and evaluate the cognitive, affective, and psychomotor domains.
- Individualize instruction.
- Allow an optimal amount of learner control.
- Assess lessons based on student objectives, attitudes, and programming effectiveness.
- Guarantee reasonable success to the student.
- Stimulate a certain degree of disequilibrium.
- Use additional media as appropriate.

Tertiary Level Design

Tertiary-level design is concerned with the incorporation of primary and secondary-level design within the context of IV. The characteristics of IV instruction that would best facilitate this process include the capacity to:

- Deal with both slow and fast learners, with endless patience.
- Provide extensive one-on-one instruction and remediation.
- Provide immediate feedback based on differing student response.

- Branch at various points, based on the students' responses to other visuals, questions, problems, etc. to deliver appropriate individualized instruction and/or feedback.
- Display text and videodisc information simultaneously
- Present and evaluate multiple choice questions as well as open-ended (essay) questions.
- Allow the student to control the pace of the lesson.
- Maintain a bug-free environment that is easy to manipulate and interpret.

Several good sources are available with additional strategies for tertiary-level design (Hazen, 1985; Kearsley & Frost, 1985; Merrill, 1988; Weller, 1988).

Hardware and Software for IV

The degree to which the above capabilities for primary, secondary, and tertiary-level design can be incorporated in a IV science lesson will depend your available hardware and software. A laser videodisc, a laser videodisc player, video monitor, and computer comprise the basic hardware (see Figure 2).

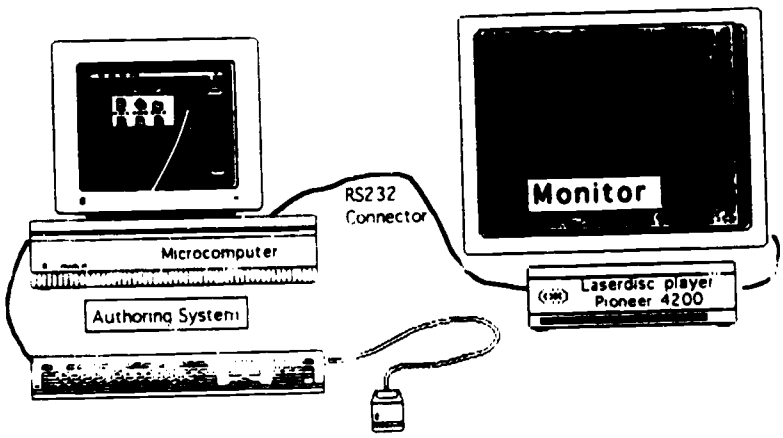


Figure 2. The components of computer-assisted IV instruction

The laser videodisc player connects to any suitable TV monitor in a similar way as the monitor would connect to a standard VCR player-recorder. While just about any computer can be linked to a videodisc player, the connection is unique. You will need to check with your local computer representative to determine the appropriate connectors and ports. A good quality laser videodisc player costs around $1000 (e.g., Pioneer L-DV 4200).

The decision of which computer to use should be directly related to your personal preference and the degree of "interactive power" you wish to ultimately attain. This, in turn, is directly related to the capabilities of available software and hardware. The software

you choose for developing your lessons depends on your programming experience. But, if you are like most teachers, with little or no programming experience, you would do best to choose one of the advanced authoring systems currently available for a variety of different microcomputers (see Tyre, 1989; Merrill, 1987; *Journal of Computer-Based Instruction*, Summer, 1984). While there are several authoring packages for the Apple IIe and IIGS, to create the IV lessons discussed earlier will require at least the power of a Macintosh or IBM Model 30/286. Hypercard for the Macintosh, and Linkway for the IBM compatibles offer easy to use iconic authoring environments.

Authoring systems contain a series of user-friendly commands, prompts, menus, and utilities that allow the user to sequence questions, videodisc content, instructional feedback, and related content in the format of a science lesson. But, beware when choosing an authoring system: some have such a rigid layout that the lesson design is pre-structured. This prevents teachers from using their *own* designs and severely limits the possibilities of non-linear instruction.

Visualization beyond IV can be achieved on the computer screen through advanced animation techniques as well as advanced graphing and drawing programs (Kashef, 1991). A different type of visualization experimentation is possible using a camcorder connected to a computer to analyze symmetry, area, and mass relationships (Speitel, 1991).

Interactive Videodisc Lesson Development—An Example

My research and development efforts at Montana State University are concerned with the development and evaluation of IV biology lessons based on the four-phase learning cycle described previously. The development process has evolved into six sequential steps: concept selection, content selection, lesson planning, authoring, evaluation, and modification. This same developmental process can be essentially followed by the classroom teacher wishing to develop IV science lessons.

Step 1- Concept Selection

Most of the major concepts in biology are suitable for learning cycle instruction. However, because of the cognitive demands and time necessary for conceptual understanding to take place, only one or two concepts should be taught per lesson. It will help to define concepts as patterns or relationships between facts, and to recognize that concepts fall on a continuum from simple to complex based on the quality and quantity of relationships or patterns between the facts. For example, the concept of diffusion is relatively simple compared to the concept of evolution. Of course, the concept(s) you choose for a lesson should depend on the students' prior knowledge, both factual and conceptual, as well as their level of cognitive development.

A recently completed IV biology lesson, "With All My Heart," deals with the concept of heart rate and its relationship to factors such as stress, age, smoking, and obesity.

Step 2- Content Selection

Once the concept has been chosen, related videodisc content is identified from a science videodisc. See Woerner, Rivers, and Vockell (1991) for a complete listing of science videodiscs. For simplicity, only the content on one side of a videodisc is used for any given

lesson since most videodisc players only read one side at a time. It is often disruptive and impractical to switch videodiscs back and forth during a lesson.

This step is facilitated by the detailed documentation provided with commercial videodiscs. The documentation describes, frame by frame, the content of the videodisc, including any audio sequences. After choosing the frame numbers, it is a simple task to splice them into the lesson at any given point or have them accessible by a specific command. "With All My Heart" involves a series of videodisc segments displaying the mammalian heart beating as a result of different physiological and environmental effects.

Ideally, content should center around activities with lengthy time periods, costly apparatus, and, in general, those experiences not available or possible in a traditional classroom.

Step 3- Lesson Planning

This task basically involves developing a detailed four-phase learning cycle lesson designed to teach the one or two science concepts. An emphasis is placed on incorporating science process skills such as observation, hypothesizing, predicting, experimenting, evaluating data, and drawing conclusions.

Based on the anticipated student responses to questions or problems posed at varying levels of difficulty, appropriate feedback is developed. This might involve remediation statements, review of earlier learning experiences, posing extra problems, as well as pointing out errors and misconceptions in students' reasoning.

A flow chart for the "With All My Heart" lesson plan is shown in Figure 3. During the "exploration" phase, students describe and record detailed observations of a living mammalian heart. Based on their observations, feedback is given that either re-directs them back to observing for more specific effects, or allows them to continue with positive reinforcement.

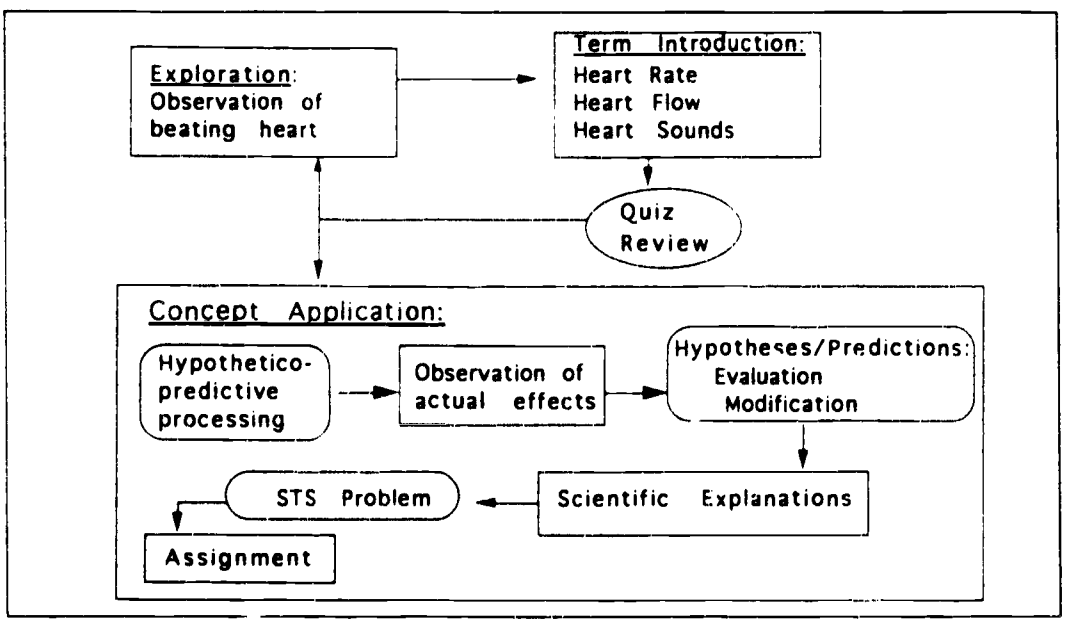


Figure 3. Flow path for IV lesson

In "term introduction," heart rate, blood flow, and heart sounds are defined using "real-live" videodisc footage in combination with computer text and graphics. Then, a short multiple choice quiz is taken with appropriate feedback provided for each possible response from the student. Following the quiz, students have the option to review the terms as many times as needed.

The "concept application" phase involves the students in making predictions, justifying predictions with logical reasons (generating hypotheses), observing actual effects of various factors on heart rate, and then evaluating and modifying the reasoning given for their initial predictions. Following the prediction evaluations, actual scientific explanations are made available to the student for further comparison and reflection. The emphasis on hypothetico-predictive reasoning facilitates students' conceptual changes. The final part of concept application involves a problem dealing with a society-technology-science (STS) issue concerning the effects of smoking and obesity on heart rate.

Step 4- Authoring

The prediction-based learning cycle lesson plans are authored for IV using "Authorware Professional" (Authorware) on a Macintosh microcomputer. In addition to possessing the capabilities for tertiary-level design, discussed earlier, this authoring system can:

- Store all students' responses, branch paths taken through the lesson, time to respond for each question, total time spent on the lesson and specific parts of the lesson, percentage score of total correct answers, etc.
- Use advanced data-driven animation (i..e, scale and coordinate movement of multiple objects) to create unique and interesting simulations of scientific phenomena.
- Use digitized sound (i.e., can speed it up and slow it down), recorded speech and recorded music.
- Be used in IBM PC's, once the lesson or course is developed.
- Jump out and back from any application (e.g., Microsoft Word, EXCEL, etc.).
- Import graphics from any application (e.g., Superpaint, MacDraw, etc.).
- Quickly design and format lessons using iconic flow-charting.
- Resume at the point in the lesson where student last finished.

The iconic environment of "Authorware Professional" makes it easy to visualize different levels of lesson design and the sequential flow of lesson development. I am currently investigating the potential of Hypercard for creating a user-friendly authoring template with which science teachers could quickly and efficiently create IV science lessons based on the learning cycle.

Step 5- Evaluation

The lesson is now field-tested with students having varying levels of science background and cognitive development. Three techniques are used to assess lesson effectiveness.

First, a selected sample of students are videotaped as they work through the lessons while thinking out loud. This technique, while involving a time consuming analysis, can yield valuable information on students' thinking processes associated with their conceptions (and misconceptions), predictions, progress through the learning cycle, and interac-

tion with IV. Readers interested in this technique should refer to Ericsson and Simon (1984) and Larkin and Rainard (1984).

Second, the computer record of student performance is analyzed for bugs, areas of confusion, etc. And third, a student questionnaire is administered with questions addressing the positives and negatives as well as suggested modifications of the lesson experience.

Step 6- Modification

Based on deficiencies identified in step 5, the computer-assisted IV lesson is modified to increase interactivity. This commonly entails considerable elaborations and additions relative to questions, questioning sequences, feedback content, and feedback location. Other modifications include additional videodisc segments, more user control, and an increased focus on personal relevancy. In general, the greater the interactivity the more intelligent the system. Recent research efforts are directed at merging IV systems with artificial intelligence. Midoro et al. (1988) remark:

> The components of the student station are changing as technology rapidly advances. As a result the potentialities of IV systems are continuously improving. Furthermore, in the near future it will also be possible to interface IV systems with AI workstations for large scale applications. Hence, the importance of an effective integration of the two technologies grows continuously. (p. 300)

It can not be denied that the above process of IV lesson development is time consuming, and it is certainly recognized the typical science teacher does not have a lot of extra time. The following suggestions to teachers will help expedite the above the six step process:

1. Purchase several good learning cycle activity books to facilitate lesson development (e.g., Carin & Sund, 1980; Lawson, 1989; Renner et al., 1985).
2. Read some interesting articles concerned with the philosophy of learning cycle teaching (Barman, 1989; Beisenherz, 1991; Lawson, 1988).
3. Modify IV lessons based on personal observations and student feedback, and questionnaire data collected during regular classroom use.
4. Train students to use an authoring system (e.g., Hypercard) so that they can assist you in authoring and modifying IV lessons. Beekman (1992) provides quick and easy to learn Hypercard instruction.

Although the think-aloud data is very valuable for lesson modification, my research has shown that questionnaire data from an entire class reveals considerable information. Further, involving students in lesson development not only improves their learning and conceptualizing, but is a powerful motivator.

Creating an IV Learning Environment in the Science Classroom

The actual logistics of IV instruction in the classroom has a wide variety of possibilities. Since most teachers will only have access to one or two IV stations, each station should be as adaptable as possible to different instruction modes. An ideal IV

learning station might involve a computer, videodisc player, monitor, a series of science videodiscs, modem, and laboratory measurement devices interfaced to the computer. This would all be housed efficiently, and safely, in a rollable table with the computer, videodisc player, and monitor on a raised platform above table level where hands-on lab activities are conducted. Interfacing scientific apparatus with the computer offers several advantages such as saving student time, improving effectiveness, simplifying data analysis, making experimental results more meaningful, and enhancing problem-solving skills (Leonard, 1988; Snyder, 1990).

Students might engage the IV learning station for individual work using science courseware or for whole-class observation and discussion. Students that are ahead of the others could be given interesting IV lessons to supplement and extend their learning while those behind could be engaged in remedial tutorials. Students, working in small groups, could use the computer for collecting, displaying, and analyzing data obtained from observations of the videodisc and/or hands-on experiences with the interfaced instruments. Immediate visual display of graphical data allows students to more easily perceive a conceptual relationship and not just a curve or line. Students could also design and conduct experiments, answer high-level questions posed, and reach collaborative conclusions. Figure 4 shows a possible cycle of scientific investigation in which technology becomes a very efficient bridge between science and mathematics while facilitating the scientific process. Both the teachers and the students will have the luxury of spending a much greater proportion of their time in science class on the "thinking" part of science investigation and learning.

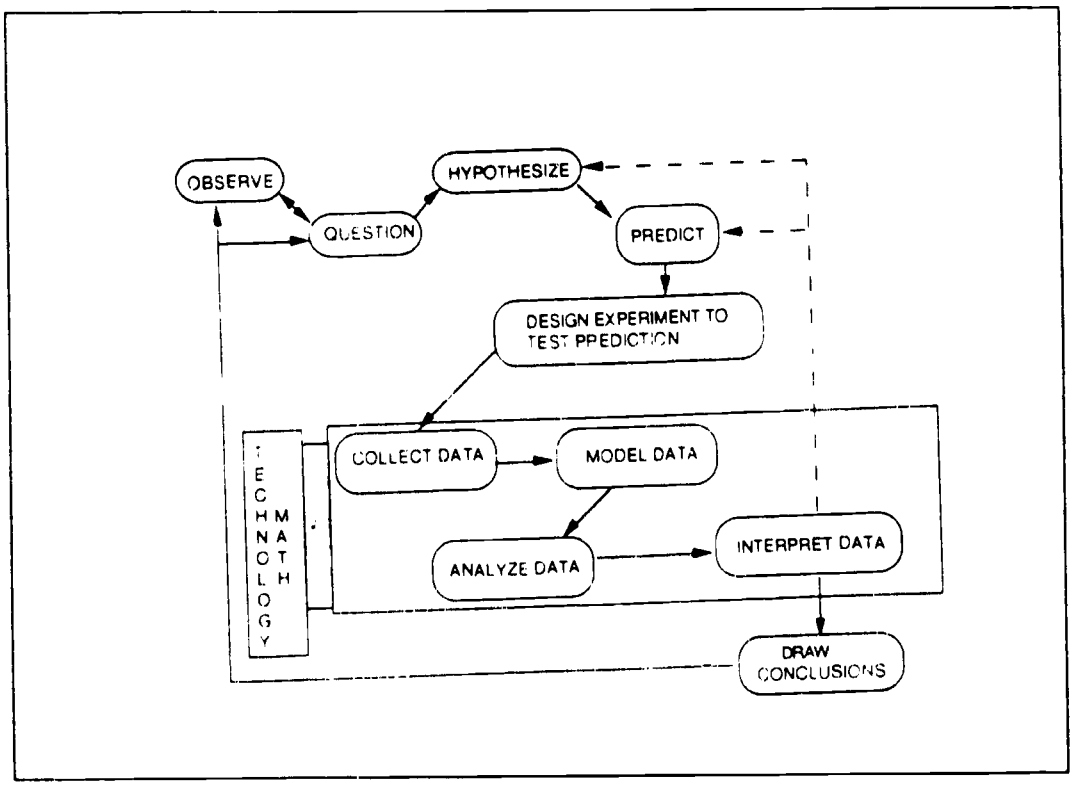


Figure 4. Cycle of scientific investigation integrating science, mathematics, and technology

BEST COPY AVAILABLE

Using the modem, teachers or students could use the IV learning station to tap into the super hi-ways of information exchange. These networks offer a variety of resources (e.g., articles, lesson plans, graphics, software, etc.) and allow the user to engage in productive collaborations with peers nation wide. While your state will have local networks that can be entered, usually for free, one of the more powerful "world" communication network is Internet. You can also enter Internet for the cost of local phone call assuming you are near a university and can get an account number and password.

It should be emphasized that the purpose of IV is not to replace traditional instructional modes, but to increase "teaching/learning power" by extending the capacity of what students and teachers can do in a science learning situation. During an IV learning session the teacher should be actively moving around the room, monitoring, answering questions, suggesting possibilities, assessing student performance, etc. Hofmeister et al. (1989) describes how a teacher might orchestrate a IV experience:

> In a typical instructional interaction, the teacher would signal the videodisc player to initiate a demonstration. The player would present the demonstration, pose problem to check on student understanding, and stop automatically, with the problem summarized on the screen. When the teacher felt the students were ready for feedback, a button press on the remote control would present the answer and often the reason for the answer. A wide range of branching options allowed the teacher to access additional examples or bypass material, based on assessments of student mastery. The use of individual student workbooks facilitated student interaction and the coordination of independent practice. In the last part of each lesson the majority of students could use workbooks for independent practice, while the teacher conducted guided practice with those in need of extra help (p. 667).

It would seem that there are many ways that IV might be used in the science classroom. The paucity of research and application relative to IV science teaching offers a simmering primordial soup anticipating the evolution of many innovative ideas and techniques which you, the science teacher, can discover.

The Future of Videodisc Technology for Science Instruction

While IV holds much promise in science education, much research is needed to examine its possibilities, successes, and failures (Reif, 1985). Clearly, future efforts for improving videodisc instruction in science should examine the extent and structure of learner interactions within the visual environment of the videodisc from a cognitive science-information-processing perspective.

> At what levels can students be involved in the design process? How are students best trained to use an authoring system so that they can assist the teacher in developing and modifying interactive videodisc lessons?
>
> How do students process information when they look at a simulation, still frame, computer graphic display? Is there a preference for motion over still picture?

Why do students choose certain learning pathways through the IV lesson, but neglect other pathways? What successful cognitive behaviors should be encouraged?

What teaching strategies are more effective for learning with IV instruction? How do students learn best from a visual display? Why are some computer displays more motivating than others? What is nature of the interaction between the computer display and the IV monitor?

What is the nature of the interaction and effects on learning for combining linguistic and visual contexts? What verbal cues should students be given and when?

What is the impact on learning of linear versus non-linear presentation formats? What are the cognitive consequences of switching from one mode to another? Does a particular mode favor concrete or formal students, field-dependent or field-independent students, slow or fast learners?

What are strategies for using IV to teach students psychomotor skills, such as those used in a science laboratory experiment?

Conclusion

The intent of this chapter was to introduce exciting teaching/learning techniques for enhancing students' thinking skills and conceptual understandings through the visually-rich medium of videodisc technology. Discussion addressed how science teachers can apply IV in their classrooms, with and without computer control, along a visually deductive to inductive continuum.

It is my hope that teachers will utilize this powerful tool to create engaging science experiences and, in general, to explore largely unexplored terrain. Teachers need to realize that they are the creators of their own destiny and, to a large degree, their students' destiny. Certainly, the in-the-trenches teachers will ultimately determine the success or failure of technology within our educational system. I believe, with the help of technology, science teachers *can* empower significant improvements in how students learn. Now more than ever, the time is right for venturing a technological leap toward a new era of teaching and learning.

References

Abraham, M. R., & Renner, J. W. (1986). The sequence of learning cycle activities in high school chemistry. *Journal of Research in Science Teaching, 23*, 121-144.

Alesandrini, K. L., & Rigney, J. W. (1981). Pictorial presentation and review strategies in science learning. *Journal of Research in Science Teaching, 18*(5), 465-474.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review, 94*, 192-210.

Anderson, J. R., Boyle, F., Corbett, A. T., & Lewis, M. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence, 42*, 7-49.

Ausubel, D. P. (1963). *The psychology of meaningful verbal learning*. New York: Grune and Stratton.

Barman, C. R. (1989, February). The learning cycle: Making it work. *Science Scope*, pp.28-31.

Basili, P. (1989). Science teaching: A matter of changing minds. *Journal of College Science Teaching, 27*, 324-326.

Beekman, G. (1992). *HyperCard 2 in a hurry*. Belmont, CA: Wadsworth Publishing Company.

Beisenherz, P. C. (1991, January). Explore, invent, and apply. *Science and Children*, pp. 30-32.

Blystone, R. V., & Dettling, B. C. (1990). Visual literacy in science textbooks. In M. B. Rowe (Ed.), *What research says to the science teacher: The process of knowing* (vol. 6). Washington, DC: National Science Teachers Association.

Blystone, R. V., & Dettling, B. C. (1990). Visual literacy in science textbooks. In M. B. Rowe (Ed.), *What research says to the science teacher: The process of knowing* (vol. 6). Washington, DC: National Science Teachers Association.

Bransford, J. D., Sherwood, R. D., Hasselbring, T. S., Kinser, C. K., & Williams, S. M. (1990). Anchored instruction: Why we need it and how technology can help. In D. Nix & R. Spiro (Eds.), *Cognition, education, and multimedia: Exploring ideas in high technology*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Bransford, J. D., Sherwood, R. D., Vye, N. J., & Rieser, J. (1986). Teaching thinking and problem solving: Suggestions from research. *American Psychologist, 41*(10), 1078-1089.

Brody, P. J. (1984). Research on pictures in instructional texts. *Educational Communication and Technology Journal, 29*, 93-100.

Bybee, R. W., Buchwald, C. E., Crissman, S., Heil, D. R., Kuerbis, P. J., Matsumoto, C., & McInerney, J. D. (1989). *Science and technology education for the elementary years: Frameworks for curriculum and instruction*. Washington, DC: National Center for Improving Science Education.

Carin, A., & Sund, R. (1980). *Discovery activities for elementary science*. Columbus, OH: Charles R. Merrill.

Derry, S. J. (1990). Learning strategies for acquiring useful knowledge. In B. F. Jones & L. Idol (Eds.), *Dimensions of thinking and cognitive instruction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Duit, R. (1991). On the role of analogies and metaphors in learning science. *Science Education, 75*(6), 649-672.

Dwyer, F. M. (1972). The effect of overt responses in improving visually programmed science instruction. *Journal of Research in Science Teaching, 9*, 47-55.

Ericsson, K., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Farragher, P. (1991, April). *What the research says about interactive video (IV) and its implications for science education*. Paper presented at the annual meeting of the National Association for Research in Science Teaching, Lake Geneva, WI.

Finke, R. (1990). *Creative imagery: Discoveries and inventions in visualization*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Fisher, K. M. (1990). Semantic networking: The new kid on the block. *Journal of Research in Science Teaching, 27*(10), 1001-1018.

Gagne, R. M., Wager, W., & Rojas, A. (1981). Planning and authoring computer-assisted instruction lessons. *Educational Technology, 21*(9), 17-26.

Glynn, S. M., Yeany, R. H., & Britton, B. K. (1991). A constructivistic view of learning science. In S. M. Glynn, R. H. Yeany, & B. K. Britton (Eds.), *The psychology of learning science*. Hillsdale, NJ: Lawrence Erlbaum.

Hannafin, M. J., & Peck, K. L. (1988). *The design, development, and evaluation of instructional software*. New York: Macmillan.

Hassard, J. (1982). Opening the mind's eye to science. *Science and Children, 19*(7), 30-32.

Hazen, M. (1985). Instructional software design principles. *Educational Technology, 25*(11), 18-23.

Hofmeister, A. M., Engelmann, S., & Carmine, D. (1989). Developing and validating science education videodiscs. *Journal of Research in Science Teaching, 26*(8), 665-677.

Holland, J., Holyoak, J., Nisbett, R., & Thagard, P. (1985). *Induction: Processes of inference, learning, and discovery*. Cambridge, MA: MIT Press.

Holliday, W. G. (1975). The effects of verbal and adjunct pictorial-verbal information in science instruction. *Journal of Research in Science Teaching, 12*(1), 77-83.

Houston, J. P. (1991). *Fundamentals of learning and memory* (4th ed.). New York: Harcourt Brace Jovanovich.

Kashef, A. E. (1991). Visualization with CAD. *Technological Horizons in Education Journal, 19*(5), 64-66.

Kearsley, G., & Frost, J. (1985). Design factors for successful videodisc-based instruction. *Educational Technology, 25*(3), 7-13.

Kozma, R. B. (1991). Learning with media. *Review of Educational Research*, 61(2), 179-211.

Larkin, J. H., & Rainard, B. (1984). A research methodology for studying how people think. *Journal of Research in Science Teaching, 21*(3), 235-254.

Lavoie, D. R. (1991, April). *Enhancing science instruction with hypothetico-predictive reasoning*. Paper presented at the annual meeting of the Montana Academy of Sciences, Billings, MT.

Lavoie, D. R., & Good, R. (1988). The nature and use of prediction skills in biological computer simulation. *Journal of Research in Science Teaching, 25*(5), 334-360.

Lawson, A. E. (1988). A better way to teach biology. *The American Biology Teacher, 50*(5), 266-278.

Lawson, A. E. (1989). *Biology: A critical thinking approach*. Tempe, AZ: Arizona State University.

Lawson, A. E., Abraham, M. R., & Renner, J. W. (1989). A theory of instruction: Using the learning cycle to teach science concepts and thinking skills. *NARST Monograph, 1*, National Association for Research in Science Teaching.

Leonard, W. H. (1988). Interfacing in the biology laboratory: State of the art. *The American Biology Teacher, 50*.

Lord, T. R. (1987). Spatial teaching. *Science Teacher, 54*(2), 32-34.

Lowery, B. R., & Knirk, F. G. (1982). Micro-computer video games and spatial visualization acquisition. *Journal of Educational Technology Systems, 11*(2), 155-166.

Mason, C. L. (1992). Concept mapping: A tool to develop reflective science instruction. *Science Education, 76*(1), 51-63.

Mayer, R. E. (1989). Models for understanding. *Review of Educational Research, 59*(1), 43-64.

Merrill, D. M. (1988). The role of tutorial and experiential models in intelligent tutoring systems. *Educational Technology, 28*(7), 7-13.

Merrill, M. D. (1987). Prescriptions for an authoring system. *Journal of Computer Based Instruction, 14*(1), 1-10.

Midoro, V., Chioccariello, A., Olimpo, G., Persico, D., Sarti, L., & Tavella, M. (1988). Interactive video and artificial intelligence: A convenient marriage. *Programmed Learning and Educational Technology, 25*(4), 299-309.

Mullis, I. V. S., & Jenkins, L. B. (1988). *The science report card: Elements of risk and recovery*. Princeton, N. J.: Educational Testing Service.

National Commission for Excellence in Education. (1983). *A nation at risk: The imperative for educational reform. Washington, D.C.* Washington, DC: U.S. Government Printing Office.

Nix, D. (1990). Should computers know what you can do with them? In D. Nix & R. Spiro. (Eds.), *Cognition, education, and multimedia: Exploring ideas in high technology*. Hillsdale, NJ: Lawrence Erlbaum.

Novak, J. D., & Musonda, D. (1991). A twelve-year longitudinal study of science concept learning. *American Educational Research Journal, 28*(1), 117-154.

Paivio, A. (1971). *Imagery and verbal processes*. New York, NY: Holt, Rinehart, and Winston.

Piaget, J. (1975). *Biology and knowledge*. Chicago, IL: University of Chicago Press.

Reed, S. K. (1992). *Cognition: Theory and applications* (3rd ed.). Pacific Grove, CA: Brooks/Cole Publishing Company.

Reif, F. (1985). Exploiting present opportunities of computers in science and mathematics education. *Journal of Computers in Mathematics and Science Teaching, 5*(1), 15-26.

Renner, J. W., Cate, J. M., Grzybowski, E. B., Atkinson, L. J., Surgber, C., & Marek, E. A. (1985). *Investigations in natural science: Biology*. Oklahoma: University of Oklahoma, Science Education Center.

Rigney, J. W., & Lutz, K. A. (1976). Effect of graphic analogies of concepts in chemistry on learning and attitude. *Journal of Educational Psychology, 68*, 305-311.

Rosenshine, B., & Stevens, R. (1986). Teaching functions. In M. C. Wittrock (Ed.). *AERA handbook of research on teaching* (3rd ed.). New York: MacMillan.

Shapiro, M. (1985, May). *Analogies, visualization and mental processing of science stories*. Paper presented at the Information Systems Division of the International Communication Association, Honolulu, HI.

Sherwood, R. D. (1991, April). *The development and preliminary evaluation of anchored instruction environments for developing mathematical and scientific thinking*. Paper presented at the annual meeting of the National Association for Research in Science Teaching, Lake Geneva, WI.

Shuell, T. J. (1990). Phases of meaningful learning. *Review of Educational Research, 60*(4), 531-547.

Simon, H. A. (1980). Information processing explanations of understandings. In P. W. Juseqyk, & R. M. Klein (Eds.), *The nature of thought: Essays in honor of D. O. Hebb*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Smith, E. E., & Lehman, J. D. (1988). Interactive video: Implications of the literature for science education. *Journal of Computers in Mathematics and Science Teaching, 8*(1), 25-31.

Snyder, D. H. (1990). Computer instrumentation and the new tools of science. *Journal of College Science Teaching, 19*(3), 171-174.

Speitel, T. W. (1991). Computer Vision. *The Science Teacher, 58*(9), 48-51.

Spiro, R. J., & Jehng, J. (1990). Cognitive flexibility and hypertext: Theory and technology for the nonlinear and multidimensional traversal of complex subject matter. In D. Nix & R. Spiro (Eds.), *Cognition, education, and multimedia: Exploring ideas in high technology*. Hillsdale, NJ: Lawrence Erlbaum.

Tyre, T. (1989). Technology update: Authoring systems. *T. H. E. Journal, 17*(3), 11-16.

Vasu, E. S., & Howe, A. C. (1989). The effect of visual and verbal modes of presentation on children's retention of images and words. *Journal of Research in Science Teaching, 26*(5), 401-407.

Weiss, I. (1989). *Science and mathematics education briefing book*. Chapel Hill, N.C.: Horizon Research, Inc.

Weller, H. G. (1988). Interactivity in microcomputer-based instruction: Its essential components and how it can be enhanced. *Educational Technology, 28*(2), 23-27.

Wheatly, G. H. (1991). Constructivistic perspectives on science and mathematics learning. *Science Education, 75*(1), 9-21.

Whitehead, A. N. (1929). *The aims of education*. New York: MacMillan.

Woerner, J. J., Rivers, R. H., & Vockell, E. L. (1991). *The computer in the science curriculum*. New York: McGraw-Hill.

Zielinski, E. J., & Sarachine, M. D. (1990). Creativity and criticism: The components of scientific thought. *Science Teacher, 57*(8), 18-22.

Chapter 5

Computer Visualization: New Window on Mathematics

DAVID A. THOMAS
MARK MITCHELL

Traditionally, mathematicians have used a variety of tables, figures, and graphs to pose and solve problems and to motivate further study. These visual devices are used most frequently to depict the given conditions of a problem or proposition and to provide a logically consistent arena complete with visual cues for exploring the relationships between the given facts and their logical consequences. In spite of these benefits, the use of figurative and graphical reasoning in mathematics education has often been limited due to a number of factors. First, few teachers or students of mathematics possess the drawing skills needed to illustrate any but the most simple mathematical objects. Second, even if a particular teacher is a gifted draftsman, the time required to construct a detailed diagram may not be warranted given the other demands placed on teachers and the limitations on class time. Third, teachers and students frequently differ in regard to the value that they attach to figurative and graphical reasoning. Some people regard figures and graphs as helpful thinking tools. Others find such devices confusing or imprecise. In spite of these difficulties, there are many examples of mathematical visualization in the traditional mathematics curriculum.

Three Examples

Figure 1 shows a unit circle centered on the origin. Given any point P on the circle, perpendiculars may be drawn from P to the x- and y-axes, the points of intersection being the coordinates of point P. A segment labeled d connects those points, forming a right triangle with hypotenuse d and right angle at P. For any given point P, what is the length of d?

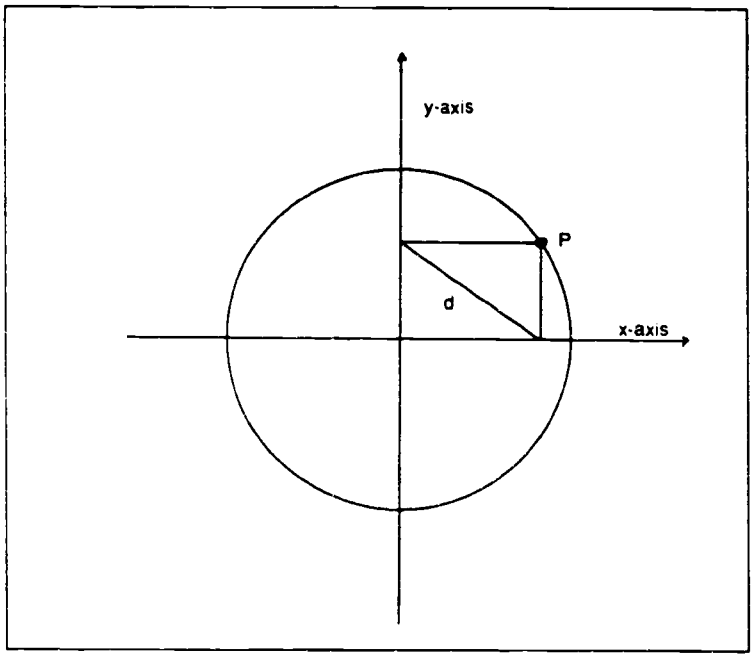


Figure 1. The length of diagonal d

Although there is an analytic solution to this problem that begins by writing an expression for segment d in terms of the coordinates of P, there is an elegant solution which is based entirely on a symmetry argument. That solution begins by observing that, by drawing perpendiculars from P to the axes, a rectangle is formed. The segment d is a diagonal of this rectangle and must be equivalent in length to the second, missing diagonal from the origin to P. That diagonal is also a radius of the unit circle. Since the diagonals are equal in length, the segment d has length 1 as well. This problem illustrates an important strategic principle: If your problem can be modeled using geometrical objects, look for a symmetry argument as the basis for a solution.

Mathematical visualization is also an important tool in concept development. For example, calculus students are often taught that the derivative of a curve at a point P may be thought of as the slope of the tangent to the curve at P. The explanation of that statement is frequently given in terms of a graph like that shown in Figure 2.

In this approach, a point P1 is chosen to the right of P and a secant is drawn passing through P and P1. Next, a point P2 is chosen between P and P1 and the same process is repeated. A third point P3 is chosen between P and P2, producing a third secant. As more and more secants are drawn, each successive secant serves as a better and better approximation of the tangent at P. The objective is for the student to form a kind of mental movie of all this that corresponds to the mathematical concept of taking a limit. In this use of mathematical visualization, the intellectual activity consists of forming a mental animation, rather than a static model, of some process. This dynamic aspect of visual memory is particularly valuable when the educational objective is to help the student form a model of a changing relationship between variables.

When students are asked to prove an assertion like the Pythagorean Theorem, constructing a suitable geometrical model which both embodies the conditions of the problem and which also suggests a solution can be very difficult. Figure 3 shows such a model. Given that QRST is a square and that each of its sides is divided into a segment of length a and a segment of length b, can you prove that $a^2 + b^2 = c^2$?

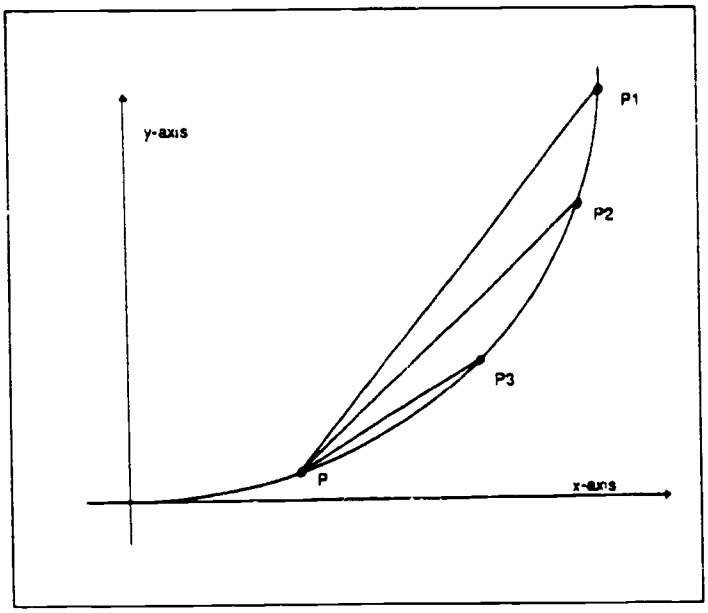


Figure 2. Approximating the slope of the tangent at P

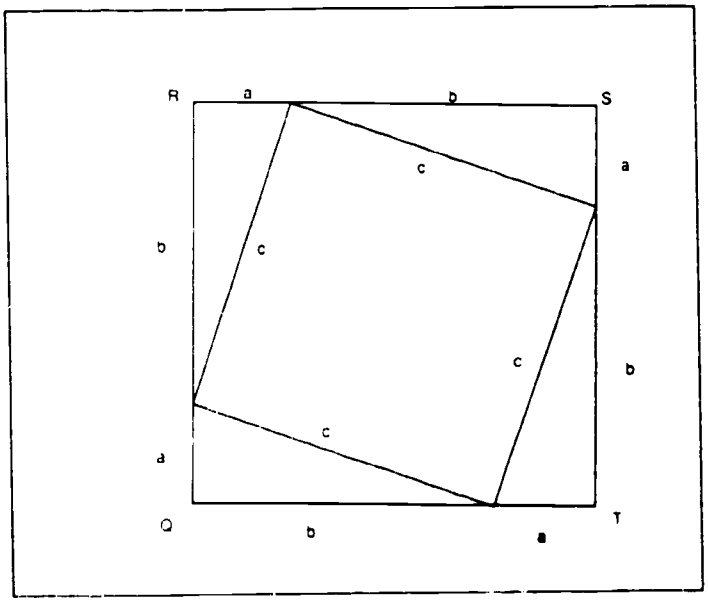


Figure 3. Proving the Pythagorean Theorem

Enlarging Our Definition of Visualization

Over the past decade, many scientists and mathematicians have adopted the use of powerful computer tools for data storage, analysis, and display. More recently, researchers with complex data analysis problems have turned to high-performance scientific visualization tools capable of showing multivariate displays, often animated over time. One consequence of this phenomenon is that a number of scientists and mathematicians are currently searching for a new definition of visualization. For example,

> Visualization...transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method of seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the ways scientists do science. (McCormick, DeFanti, & Brown, 1987)

Focusing more directly on the issue of visualization in mathematics, Zimmerman and Cunningham (1991) state that

> Vision is not visualization; to see is not necessarily to understand. In mathematics, visualization is not an end in itself but a means toward an end, which is understanding. If mathematics is the science of patterns, it is natural to try to find the most effective ways to visualize these patterns and to learn to use visualization creatively as a tool for understanding.

These statements may not qualify as definitions, but they do identify computer visualization as a powerful new tool in the service of understanding. To further develop this theme, three examples are presented which illustrate the value of computer visualization as a tool for concept development in high school and undergraduate mathematics. Both existing and new technologies will be considered.

Example 1: Fractals, Logo, and Traditional Mathematics

During the 1980's, the study of dynamical systems emerged as one of the fastest growing branches of mathematics. Thanks to the efforts of Benoit Mandelbrot (1977, 1982), Heinz-Otto Pietgen (1986, 1988), Michael Barnsley (1988), James Gleick (1987), and a number of other researchers and writers, a fascinated public was introduced to the branch of dynamical systems called fractal geometry. Although the pedigree of fractal geometry is still being argued by professional mathematicians, the computer science community has taken this new tool and used it to dazzle the public with spectacular color graphics, including realistic animations of alien worlds.

Like many other branches of mathematics, fractal geometry may be investigated at several levels. This example introduces the concept of deterministic, self-similar fractals by using a familiar tool to many elementary and secondary school teachers, Logowriter, and by making use of a number of mathematical concepts taught at the high school level, typically in Algebra II. The focus of the illustration is the von Koch snowflake shown in Figure 4.

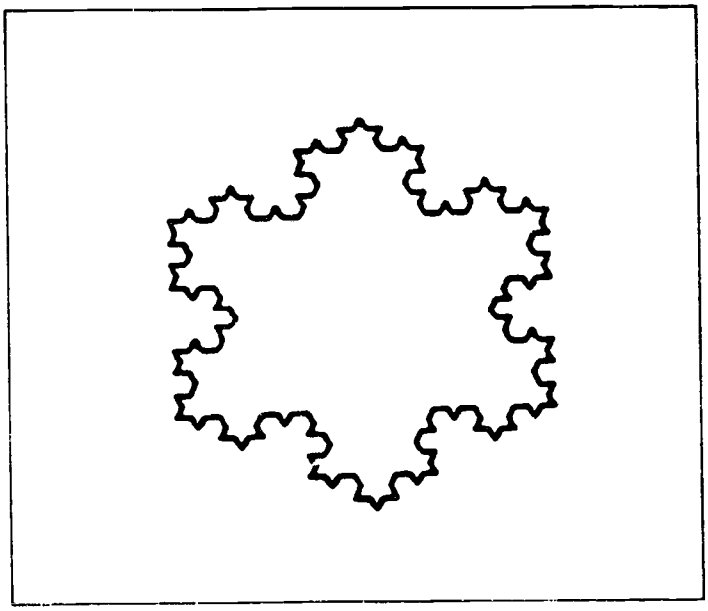


Figure 4. Snowflake curve

The most significant characteristic of the snowflake curve shown in Figure 4 is its self-similarity. Objects that are self-similar repeat some geometrical theme on different scales with the following result: small pieces of the whole strongly resemble larger pieces. This quality of self-similarity is one of the attributes most often associated with the objects commonly called fractals.

Creating self-similar objects is really quite simple in principle. In the case of the von Koch snowflake curve, the construction proceeds as follows. Beginning with an equilateral triangle (Figure 5), the center third of each segment of the figure is removed and replaced with two congruent segments arranged as an equilateral "bump-out." This produces a star like that shown in Figure 6. This star has six points and therefore twelve segments. Each of these segments is then divided into thirds. The center third is removed and replaced by small equilateral bump-outs as shown in Figure 7. Repeating this theme on smaller and smaller scales, the snowflake curve develops the intricate perimeters shown in Figures 4 and 8

Since constructing such objects with pencil and paper is time consuming, few mathematicians and fewer mathematics teachers and students over the years have bothered to explore these types of objects. The general availability of computers and software packages like Logowriter now make this task much easier and far less time consuming. Since very few people are capable of imagining the final form of a fractal, the availability of computer visualization tools is a critical factor in bringing this type of mathematics into the classroom.

Using the Logowriter procedures in Program Listing 1, it is a simple matter to generate the von Koch snowflake. For example, the Logowriter procedures collected in Program Listing 1 draw the von Koch snowflake in a matter of seconds. Figure 5 is obtained using the command DO 0, Figure 6 by DO 1, Figure 7 by DO 2, and Figure 8 by DO 3.

ERIC

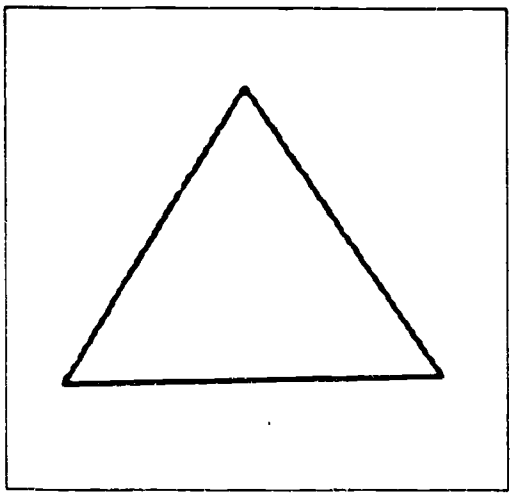


Figure 5. Equilateral triangle

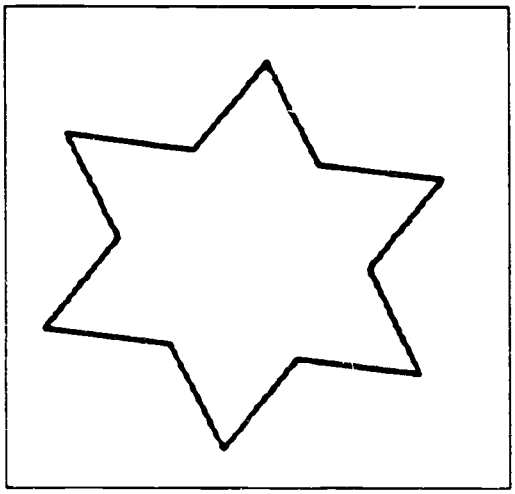


Figure 6. Six-pointed star

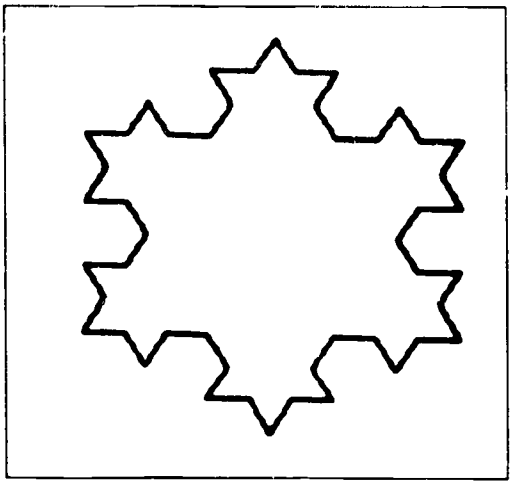


Figure 7. Two-levels of iteration

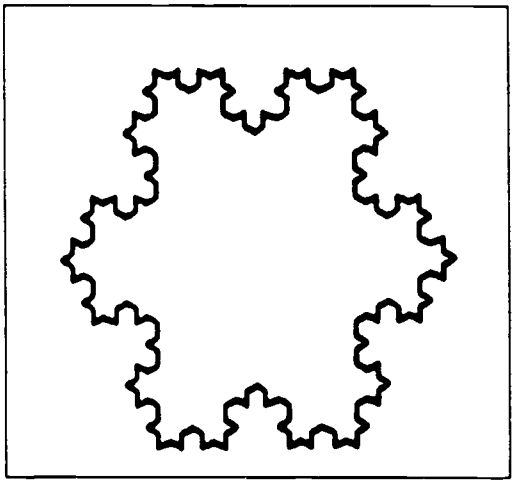


Figure 8. Three-levels of iteration

Program Listing 1

```
TO DO :N
START
REPEAT 3 [IFELSE :N=0 [FD :L] [LINE :N] RT 120]
END

TO START
MAKE "X 1
REPEAT :N [MAKE "X 3*:X]
MAKE "L 162/:X
RG PU HT RT 60 BK 90 LT 30 PD
END
```

```
TO LINE :Y
IFELSE :Y=1 [SMALLEST] [STEPDOWN]
END

TO SMALLEST
FD :L LT 60 FD :L RT 120 FD :L LT 60 FD :L
END

TO STEPDOWN
LINE :Y-1 LT 60 LINE :Y-1 RT 120 LINE :Y-1 LT 60 LINE :Y-1
END
```

Naturally, by changing the shape of the original object from an equilateral triangle to some other regular polygon, or by changing the shape of the bump-out, a different fractal will be produced. A discussion of how to do so is found in Thomas (1989).

In presenting the von Koch snowflake to a group of high school students, the following thoughts may be used to provide a direction for class discussion.

1. After demonstrating the evolution of the curve and discussing the term fractal, ask
   * As we increase the parameter controlling the number of levels at which the object's "theme" is expressed, what happens to the perimeter of the object?
   * What happens to the area of the object?
   * If you could DO (infinity), what would the perimeter be? What would the area be?
2. To examine what happens to the perimeter, collect data and look for a pattern.
   * First count the number of segments at each "level" and record the data in a table.

n	# of segments	
0	3	= 3
1	4(3)	= 12
2	$4^2(3)$	= 48
3	$4^3(3)$	= 192
n	$4^n(3)$	

* Next observe that the length of each segment may be written as $L/(3^n)$, where L is the length of the side of the original equilateral triangle.
* You may then write the perimeter of the curve as
(# segments)(segment length) = $(4^n(3))(L/3^n) = 3L(4/3)^n = P_0(4/3)^n$, where P_0 is the perimeter of the original equilateral triangle. Clearly, as n increases without bound, the perimeter does the same. This is an example of a divergent geometric sequence.

ERIC

3. To find out what happens to the area, follow the same approach.
   *Create a table of data based on the triangular areas shown in Figure 9.

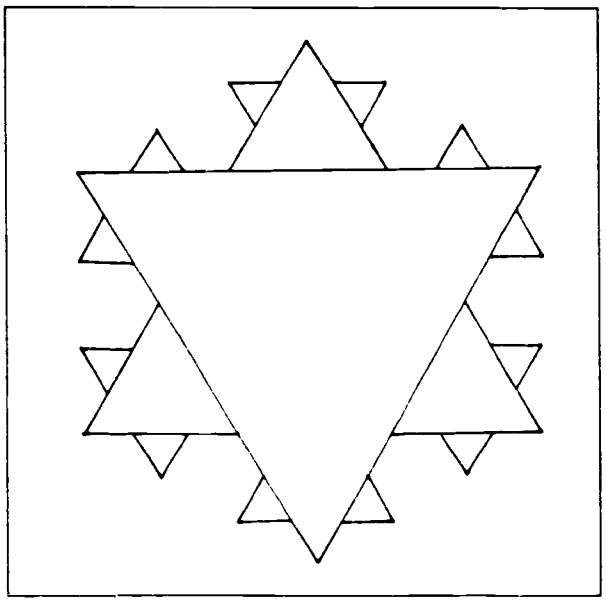


Figure 9. Building a snowflake boundary

n	s	# triangles added = # of segments at the previous level	additional area
0	3		
1	12	3	$3(1/9)$
2	48	12	$12(1/9)^2$
3	192	48	$48(1/9)^3$
n	$4^n(3)$	$4^{n-1}(3)$	$4^{n-1}(3)(1/9)^n$

If the original equilateral triangle has an area of 1, the area for the curve on the n-th level of iteration is given by

$$1+\sum_{i=1}^{n} 4^{i-1}(3)\left(\frac{1}{9}\right)^i = 1+\frac{1}{3}\sum_{i=1}^{n}\left(\frac{4}{9}\right)^{i-1}$$

which, as n goes to infinity, is a convergent geometric series with sum 1.6.

4. Summarizing the findings, the von Koch curve in its limiting case has an infinite perimeter but a finite area.

Additional connections to the secondary school mathematics curriculum may be developed by investigating the concept of fractal dimension (Thomas, 1989). In the case of the von Koch snowflake, the fractal dimension is obtained by computing

$$(\log 4) / (\log 3) = 1.261859507$$

where 4 segments replace 3 segments in the course of dividing a side of the polygon into thirds, removing the center third, and replacing it with an equilateral bump-out consisting of 2 segments.

By modifying Program Listing 1, it is possible to create a wide variety of self-similar objects. Figures 10-12 show the first few steps in generating three other self-similar objects. Each of these fractals has rotational and line symmetries which could serve as the focus of a group discussion. The Logowriter code for these fractals is found in Program Listings 2-4. Students who are familiar with Logowriter and who understand recursion can modify these program listings to produce many more self-similar objects.

Investigations such as the one just discussed cannot take place without appropriate computer visualization tools. But having the tools on hand doesn't necessarily mean that teachers and students will use the tools to expand or extend their understanding of mathematics. Both teachers and students must learn to think visually as well as symbolically. For some individuals, informal mathematical investigation of fractals may provide an entertaining context for the development of these traits.

Example 2: Introducing Linear Transformations Using Logowriter

The study of matrix algebra has long been a part of the undergraduate mathematics curriculum. And because matrix algebra is used extensively in many branches of science, engineering, and business, most students majoring in those disciplines take courses in which matrix algebra is used. Unfortunately, the traditional high school mathematics curriculum has done little if anything to prepare its college bound students for college level matrix algebra. In recognition of this fact, the NCTM's *Curriculum and Evaluation Standards for School Mathematics* (Working Groups of the Commission on Standards for School Mathematics, 1989) calls for the introduction of matrix algebra at the high school level. Consistent with the *Standards'* general goal of making mathematics a meaningful activity for students and specifically related to the *Standards'* emphasis on mathematical connections and transformation geometry, this example illustrates a visual approach to the introduction of linear transformations.

Using functional notation, a linear transformation T may be represented as $T(x,y) = (x',y') = (ax + by + c, dx + ey + f)$, where (x,y) is a given point in the coordinate plane before transformation and (x',y') is the image of the given point after transformation. In matrix notation, this expression may be written using a 2 X 2 scaling matrix and a 2 X 1 translation matrix as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ab \\ de \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

BEST COPY AVAILABLE

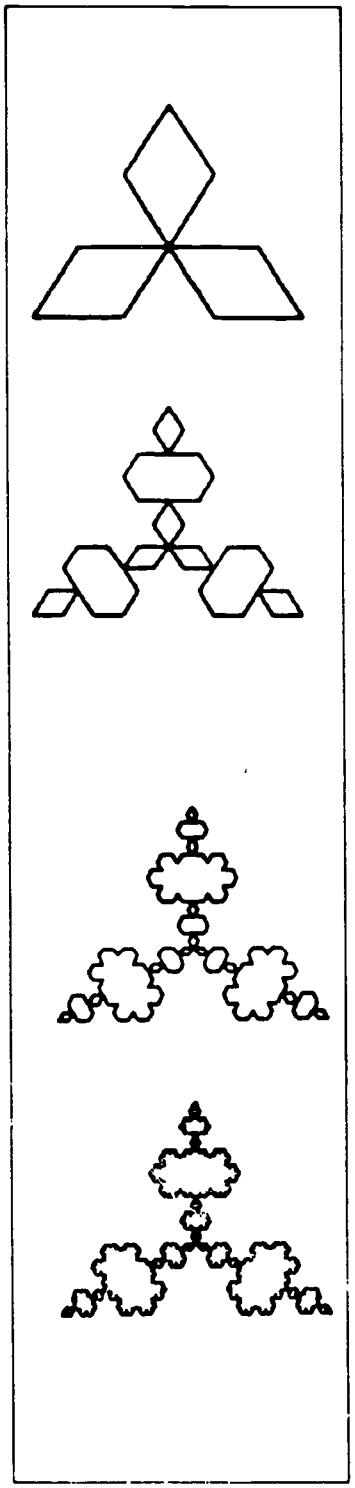


Figure 10. Variation 1

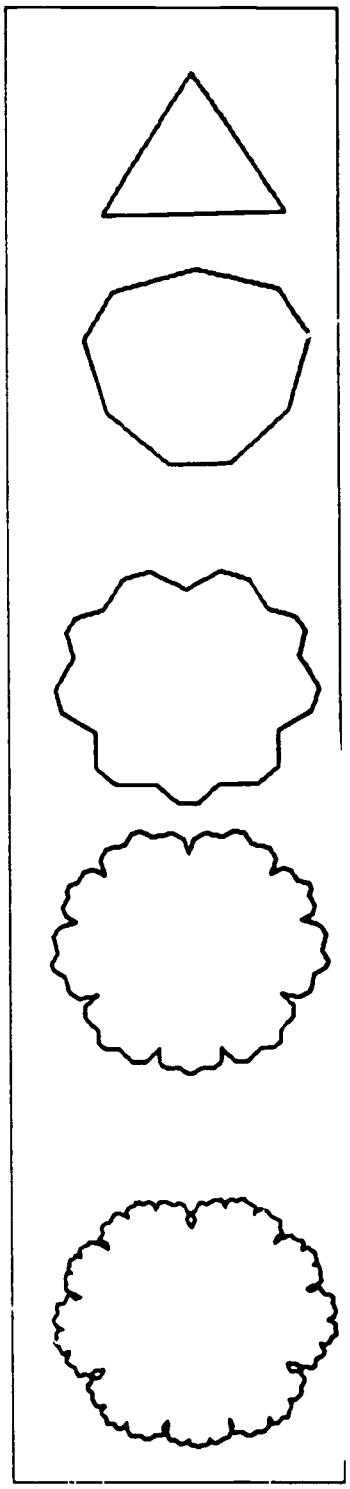


Figure 11. Variation 2

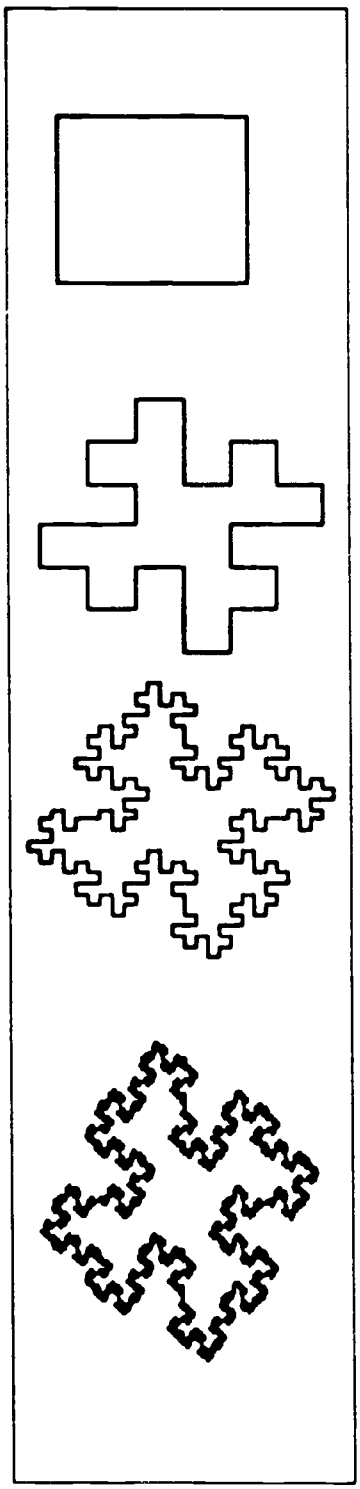


Figure 12. Variation 3

Program Listing 2

```
TO DO :N
START
REPEAT 3 [IFELSE :N=0 [FD :L] [LINE :N] RT 120]
END

TO START
MAKE "X 1
REPEAT :N[MAKE "X 3*:X]
MAKE "L 162/ :X
RG PU HT RT 60 BK 90 LT 30 PD
END

TO LINE :Y
IFELSE :Y=1 [SMALLEST] [STEPDOWN]
END

TO SMALLEST
FD :L RT 60 FD :L LT 120 FD :L RT 60 FD :L
END

TO STEPDOWN
LINE :Y-1 RT 60 LINE :Y-1 LT 120 LINE :Y-1 RT 60 LINE :Y-1
END
```

Program Listing 3

Copy procedures TO DO, TO START, and TO LINE friin Program Listing 2, and then add:

```
TO SMALLEST
LT 45 FD :L*1.414 RT 45 FD :L RT 45 FD :*1.414 LT 45
END

TO STEPDOWN
LT 45 LINE :Y-1 RT 45 LINE :Y-1 RT 45 LINE :Y-1 LT 45
END
```

Program Listing 4

```
TO DO :N
START
REPEAT 3 [IFELSE :N=0 [FD :L] [LINE :N] RT 120
END

TO START
MAKE "X 1
REPEAT :N[MAKE "X 4*:X]
MAKE "L 108/:X
RG PU HT RT 60 BK 50 LT 30 PD
END

TO LINE :Y
IFELSE :Y=1 [SMALLEST] [STEPDOWN]
END

TO SMALLEST
FD :L LT 90 FD :L RT 90 FD :L RT 90 FD :L FD :L
LT 90 FD :L LT 90 FD :L RT 90 FD :L
END

TO STEPDOWN
LINE :Y-1 LT 90 LINE :Y-1 RT 90 LINE :Y-1 RT 90 LINE :Y-1 LINE :Y-1
LT 90 LINE :Y-1 LT 90 LINE :Y-1 RT 90 LINE :Y-1
END
```

A better approach is to use a single 3 X 3 matrix which combines both the scaling and translation operations.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The expression for T(x,y) shows that the x- and y-coordinates of the image point (x',y') are formed using a linear combination of the x- and y-coordinates of the given point. It is also clear that the transformation is defined for any choice of (x,y). Although this definition may be accurate, it does not provide any geometric sense of the consequences of applying the transformation to a given point, line, or other object in the plane. Supplementing this analytic definition of a linear transformation with a visual model improves the situation considerably.

Visually, a linear transformation can be thought of as a function that takes rectangles as an input and returns parallelograms as an output. Under linear transformations, straight lines remain straight, though their length and direction may change. Another consequence is that parallel lines remain parallel. Lines that intersect before also intersect after transformation, though the angle formed by the two lines may

change. Added to the previous definition, the concept of a linear transformation suddenly takes on the reality of a visual model. For many students, this in itself is enough to motivate further study into the nature of linear transformations. Logowriter Program Listing 5 provides a limited opportunity to do so. The limitation occurs in that the only transformations allowed are those that take a rectangle as an input and return a rectangle as an output. No rotations or shears are permitted so as not to overly complicate the student's first experience with linear transformations.

Program Listing 5

```
to do :n :w :h :xshift :yshift
rg ht
ref
make "xscale 1
make "yscale 1
make "ax -80
make "ay  80
shrink :n
end

to ref
pu setpos [-80  80] pd rt 90
repeat 4 [fd 160 rt 90]
end

to shrink :n
repeat :n [make "xscale :w*:xscale
make "yscale :h*:yscale
make "bx (:w*:ax + :xshift)
make "by (:h*:ay + :yshift)
pu fd (:bx - :ax) lt 90 fd (:ay - :by) lt 90 pd
repeat 2 [fd :xscale*160 rt 90 fd :yscale*160 rt 90]
make "ax :bx
make "ay :by
show pos]
end
```

The syntax for using the LINTRAN procedures is as follows:

DO (# reps) (x-scale) (y-scale) (x-shift) (y-shift)

where (# reps) is the number of times that the transformation is to be applied to the starting rectangle. Use 1 initially.

(x-scale) is the factor by which the horizontal dimension of the starting rectangle is to be shortened or lengthened. Use .5 initially.

ERIC

(y-scale) is the factor by which the vertical dimension of the starting rectangle is to be shortened or lengthened. Use .5 initially.

(x-shift) is the horizontal amount by which the transformed rectangle is to be shifted.

(y-shift) is the vertical amount by which the transformed rect angle is to be shifted.

Taking care to insert spaces between the parameters, a first use of the LINTRAN procedures might begin with the command

DO 1 .5 .5 -10 5

A more interesting result will be obtained by repeatedly applying the same transformation to the plane, resulting in a shrinking set of rectangles. For example, if you try DO 10 .8 .7 -1 5 you obtain the image seen in Figure 13.

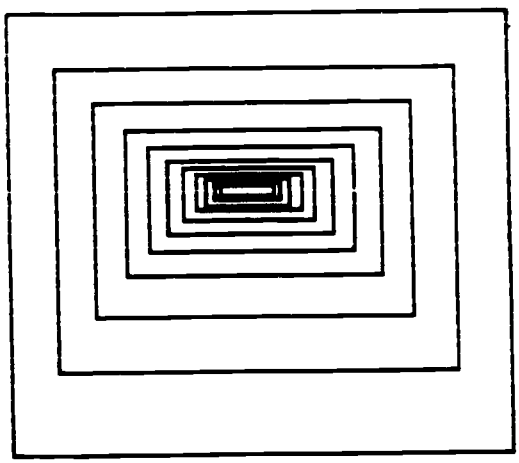


Figure 13. Fixed points and contractive affine maps

After a little experimentation with this program it is not unusual for students to observe that for certain linear transformations, repeated applications of the transformation shrink the starting rectangle to a point on the computer screen. A more subtle observation is that this point may be located anywhere in the plane, not just inside of the starting rectangle. Another observation of interest is that this process represents composing the transformation with itself over and over. These observations may be used as a motivation for introducing the concept of fixed points.

Figure 14 shows a unit square on an inverted coordinate system such as is found on computer screens. The corners of the square are {(0,0), (1,0), (0,1), (1,1)}.

If a linear transformation matrix T is defined as

$$T = \begin{bmatrix} 1/2 & 0 & 7/16 \\ 0 & 1/2 & 1/4 \\ 0 & 0 & 1 \end{bmatrix}$$

then any point which does not move under the translation will be a solution to the matrix equation

$$\begin{bmatrix} 1/2 & 0 & 7/16 \\ 0 & 1/2 & 1/4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

In this case, the fixed point is (7/8, 1/2). This is the only point in the plane which does not move under the transformation. And, since the transformation is a contraction mapping, moving points closer together, every point on the plane moves closer to the fixed point with successive applications of T. For this reason, fixed points of this sort are called attractors. If you can run this process as a mental movie, you should imagine the entire plane being drawn into the attractor as if the fixed point were a kind of mathematical black hole.

From considering one linear transformation and its fixed point, we now move on to consider a system of three linear transformations $\{T_1, T_2, T_3\}$ where

$$T_1 = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_3 = \begin{bmatrix} 1/2 & 0 & 1/4 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

Each of these transformation is a contraction mapping with x-scale and y-scale factors of 1/2. The three transformations differ in their fixed points (0,0), (1,0), and (1/2,1) respectively.

Under each of these transformations, the starting unit square would be mapped to a smaller square as shown in Figure 15. Under the first transformation, the unit square is mapped to the shaded region marked T_1. Similarly, the second and third transformations would map the unit square to regions T_2 and T_3 respectively. Note that no region of the original unit square would be mapped to the unshaded part of Figure 15. It is also clear that the shaded regions occupy 3/4 of the original area of the unit square. These shaded regions show where a given point in the original unit square might end up after one transformation of the unit square by one of the three linear transformations.

We now consider the composition of two transformations, T_i and T_j and ask the question, "Where might the image of a point in the original unit square end up after it undergoes two successive mappings?" Figure 16 shows nine shaded regions, each a different ordered pair of transformations. Figure 16 also illustrates the fact that matrix multiplication is typically not commutative and that the shaded region now occupies 9/16 of the area of the original unit square.

Following this same line of reasoning, Figure 17 shows the result of applying three successive transformations to the unit square. The composition of three transformations produces 27 possible regions where an image point may be found, constituting 27/64 the area of the unit square.

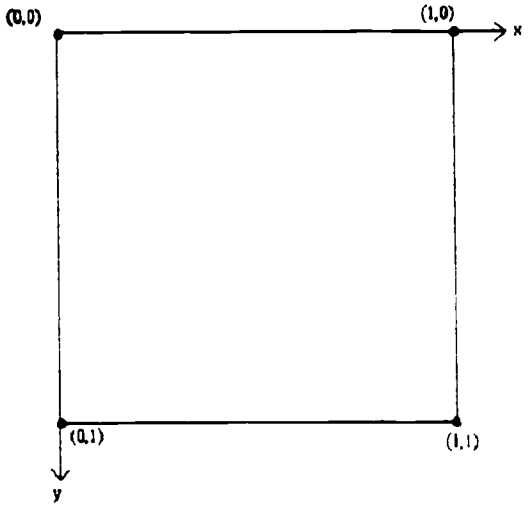


Figure 14. Typical computer screen coordinates

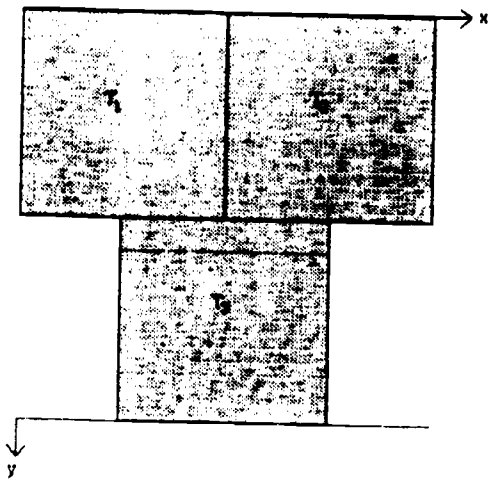


Figure 15. After one iteration of the IFS

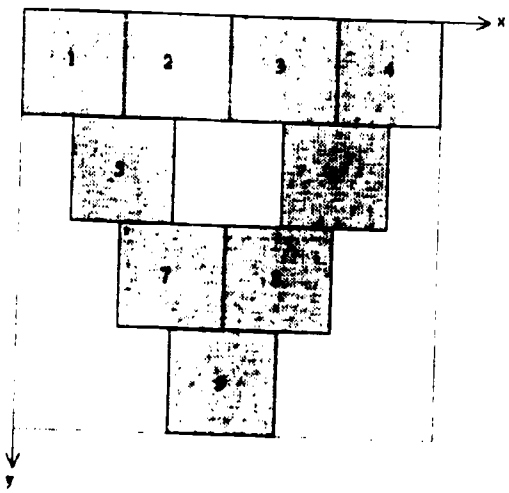


Figure 16. After two iterations of the IFS

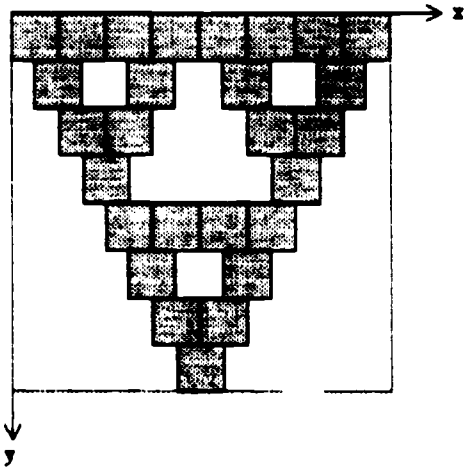


Figure 17. After three iterations of the IFS

What happens as we continue to apply more and more transformations? In general, after n transformations, the shaded portion of the figure will consist of 3^n regions occupying $(3/4)^n$ of the unit square. As n approaches infinity, the number of regions goes to infinity while the area occupied by those regions goes to zero. The theoretical object that emerges from this process as n goes to infinity is an exquisitely detailed set of points called the Sierpinski gasket. Figure 18 shows the general form of this object.

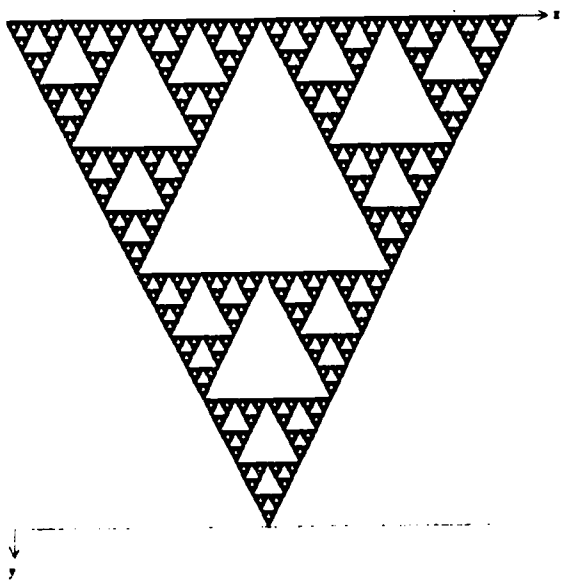


Figure 18. After many iterations of the IFS

The process just described can be modeled quite easily using the set of Logowriter procedures found in Program Listing 6. To run the program, simply give the following commands: DO 0; DO 1; DO 2; etc.

Program Listing 6

```
to do :n
rg ht
pu setpos [-100 120] pd
exp2 :n
make "s 160/:x
ifelse :n > 0 [tribox :n]
[fd 160 rt 90 fd 160 rt 135 fd 160*1.4142 rt 135 pu fd 15 rt 90 fd 5 pd fill ]
end

to exp2 :n
make "x 1
repeat :n [make "x 2*:x]
end
```


ERIC

```
to box :n
repeat 2[fd :s rt 90]
rt 45 fd :s*1.4142 rt 160
pu fd 3 pd fill pu bk 3 lt 25 pd
end

to tribox :n
ifelse :n = 1 [box :n fd :s box :n rt 90 fd :s lt 90
box :n rt 45 bk :s*1.4142 lt 45]
[make "x :x/2 tribox :n-1 fd :s*:x tribox :n-1 rt 90 fd :s*xlt 90
tribox :n-1 rt 45 bk :s*1.4142*:x lt 45 make "x :x*2]
end
```

Clearly, the Sierpinski gasket fits our definition of a self-similar object. What is most astonishing about the whole exercise, however, is that a system of three linear equations having three fixed point attractors ends up generating a "strange" attractor for the system with consists of an infinite number of points, the Sierpinski gasket. What is even more astonishing is that a very simple procedure exists for generating an approximation for the strange attractor of such systems. Michael Barnsley (1988) calls it the Chaos Game.

Given any system of contractive linear transformations, the Chaos Game proceeds as follows. A random seed point is selected in the plane. One of the linear transformations of the system is randomly selected and the seed point's image is determined and its location plotted. The coordinates of this point are then used as the input for a second randomly selected linear transformation. The entire process is repeated thousands of times, generating a set of several thousand points on the computer screen.

It is a simple matter to write a BASIC program such as found in Program Listing 7 to mimic this process. If you run that program, you will get an object that looks like a Christmas tree (see Figure 19).

Program Listing 7

```
1LIST        2RUN        3LOAD"        4SAVE"        5CONT←
10 SCREEN 1
100  X = 0:Y = 0
110 I = INT (4*RND (1) + 1)
120 IF I = 1 THEN WX = .6*X + .18 : WY = .6*Y + .35
130 IF I = 2 THEN WX = .6*X + .18 : WY = .6*Y + .12
140 IF I = 3 THEN WX = .4*X  + .3*Y + .27:WY = -.3*X + .4*Y + .36
150 IF I =4 THEN WX = .4*X -.3*Y + .27:WY = .3*X + .4*Y + 9.000001E-02
160 X = WX : Y = WY
170 PX = INT (X*250) : PY = INT ( (1-Y)*250)
180 PSET (PX, PY)
190 GOTO 110
```

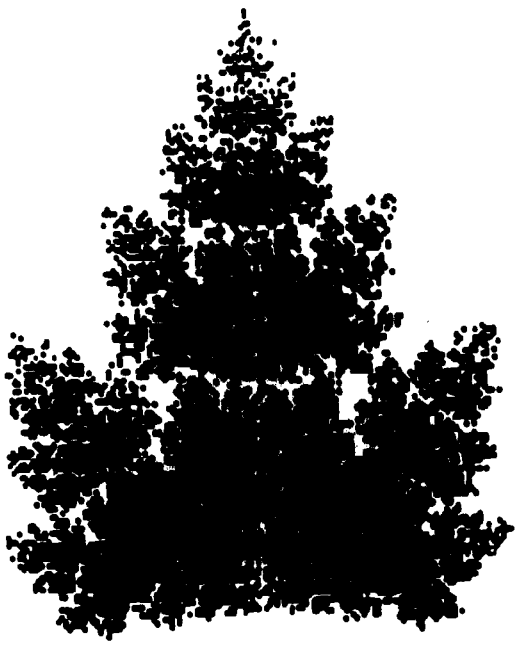


Figure 19. Fractal leaf

If you use this approach to iterate the set of transformations T_1, T_2, and T_3 given above, you will obtain the same image generated by the Logowriter procedures given in Program Listing 6. That surprising result is explained by the fact that after 8-10 iterations, the regions determined by the transformations are smaller than the pixels on the computer screen. It doesn't matter whether you are plotting points or regions. After a few iterations, both results are indistinguishable on your computer screen. Under such circumstances, it is far more efficient to plot points rather than regions of the plane.

Another surprising result is that no matter how many times you run the Chaos Game, the images generated always look the same, even though the random selection of transformations leads to a different sets of points for each image. The explanation for this astonishing result is that both images are approximations of the same theoretical entity, called the "strange attractor" of the iterated function system (IFS). How can this be? It's like pea plants all grown from the same pod of seeds. They may differ in a few details, but given basically the same sun, water, and soil, they will turn out remarkably similar. The strange attractor of an IFS system corresponds to the pea plant you'd get if it had the ideal amount of sun and water and a perfect soil in which to grow.

Here is a mathematical metaphor for growth in the natural world. You are an approximation of the strange attractor determined by your DNA! Every day of your life, your cells repeat the same set of operations. Sure, if you'd had more vitamins, you might have grown taller. And if you'd never had that childhood disease you might have been stronger. But you would still be YOU, only better.

You may be able to guess a few of the broad features of an attractor before it is created, but you can never know the specific points that will be produced along the way.

The unpredictable nature of the random process used to generate those points guarantees that you can never see the final result without actually going through all the intermediate steps. To an observer, the process is chaotic even though the result is creative.

Here then is an aspect of mathematics for which computer visualization is indispensable. Students can define IFS systems, run them on a computer, and watch the strange attractors materialize on the screen. With a little practice, strange attractors can be designed to have particular characteristics. Two particularly helpful tools for this type of work are the *Desktop Fractal Design System* and *Chaos: The Software.*

The first two examples of computer visualization have focused on topics related to fractal geometry, a relatively new branch of mathematics. In both cases, the technology required has been simple, a PC and a copy of Logo. The third example deals with a new technology and an old topic, number theory.

Example 3: Exploring Modular Arithmetic

Think back to when you first learned how to divide one whole number by another, say 21 divided by 5. You were probably taught to write your answer as a quotient plus a remainder, in this case 4 R 1. Later on, you abandoned remainders as fractions and decimals took over the job of describing the "leftovers" typically obtained in division problems. This example picks up where remainders left off by introducing the notion of modular arithmetic.

Returning to the problem 21 divided by 5, we can define the expression 21 (mod 5) to mean the remainder obtained when 21 is divided by 5. In this case, we can write 21 (mod 5) 1, read "twenty-one mod 5 is congruent to 1." There are many other whole numbers n congruent to 1 (mod 5). In general if $n = 5k + 1$, where k is a whole number, then n (mod 5) 1. For example, the numbers 6, 11, 16, 21, 26, and so on are all congruent to 1 (mod 5). The set of all whole numbers n congruent to 1 (mod 5) is called the congruence class [1]. There are four other (mod 5) congruence classes corresponding to remainders of 0, 2, 3, and 4. The names of these congruence classes are [0], [2], [3], and [4]. As defined above, each congruence class consists of infinitely many whole numbers, all of which yield the same remainder when divided by 5. It is clear, however, that each congruence class has a smallest non-negative member. For instance the smallest member of [3] is 3. The technical term for the smallest non-negative member of a congruence class is the least residue of the congruence class. This example is specifically concerned with the least residues associated with the congruence classes produced in a variety of problems involving modular arithmetic.

Now that it is clear how an arithmetic expression such as 21 (mod 5) is to be evaluated, it is natural to extend this concept to include algebraic expressions such as x (mod 5), where x is any whole number. The only possible answers belong to the five congruence classes [0], [1], [2], [3], and [4]. In this sense mod can be thought of as an operator like multiplication or division, producing a graphable result. What would a graph of this look like? As Figure 20 suggests, the graph of x (mod 5) consists of discrete points arranged in a sawtooth pattern. This pattern is even more pronounced (see Figure 21) when we redefine x to be any positive real number. Figure 21 shows that the height of the graph is determined by the size of the modulus, in this case 5. Choosing a larger modulus produces a higher graph, i.e. a graph with longer teeth.

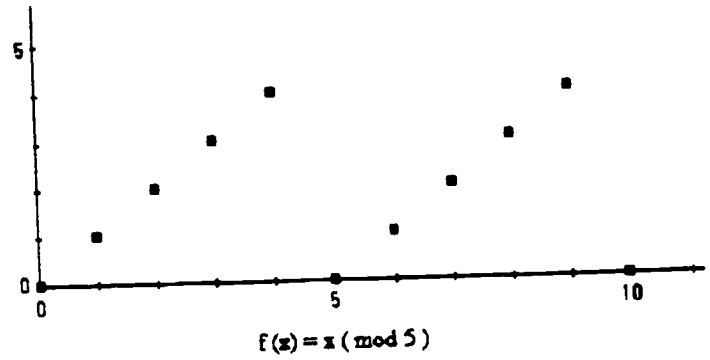


Figure 20. f(x) = x (mod 5), x ∈ Non-negative Integers

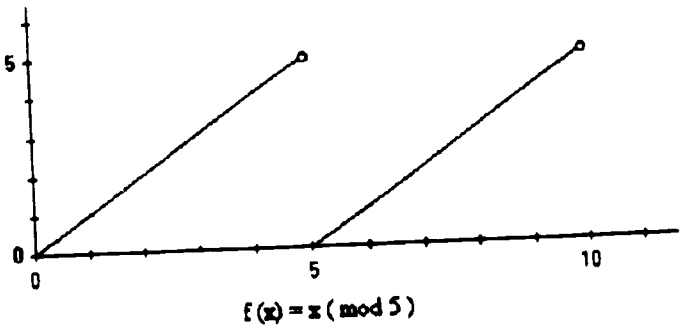


Figure 21. f(x) = x (mod 5), x ∈ Non-negative Real Numbers

The expression 2x (mod 5) is graphed in Figure 22. Multiplying the variable x by 2 has the effect of shortening the base of each tooth by one-half because the expression runs through the congruence classes twice as fast as for x (mod 5). This is similar to the effect observed when graphing the trigonometric functions: doubling the coefficient of the angle halves the period. In general, the graph of the expression Ax (mod C) will be a sawtooth curve with period C/A and amplitude C. Finally, as with the trigonometric functions, the introduction of a constant B in the expression Ax + B (mod C) produces a phase shift in the graph of Ax (mod C). Figure 23 shows this in the case of the graph of 2x + 3 (mod 5). In general, the amount of phase shift is given by B/A.

At this point, we can consider more complicated expressions such as Ax + B (mod Cx + D) in which the modulus of the expression is not constant but is itself a function of x. The introduction of the terms Cx + D in the expression for the modulus forces us to abandon the notions of a constant period and phase shift. Graphically, this results in teeth of varying width and height. Figure 24 shows the graph of the expression 7x + 2 (mod 3x + 5). Relating the expression to the graph and remembering that the amplitude of any tooth is limited by the value of the modulus at that point, it is not surprising that the teeth appear to shrink in size as they approach the point x = 5/3, the solution to the equation 3x + 5 = 0.

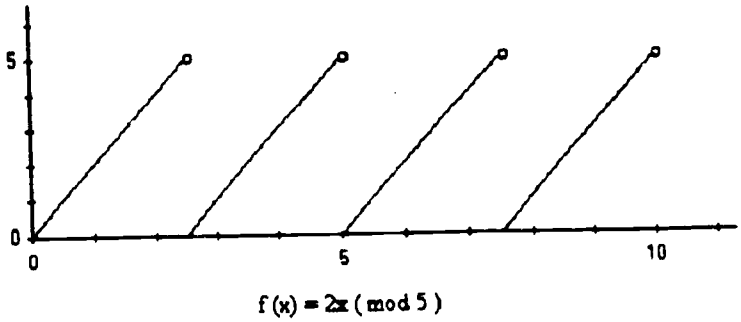


Figure 22. f(x) = 2x (mod 5)

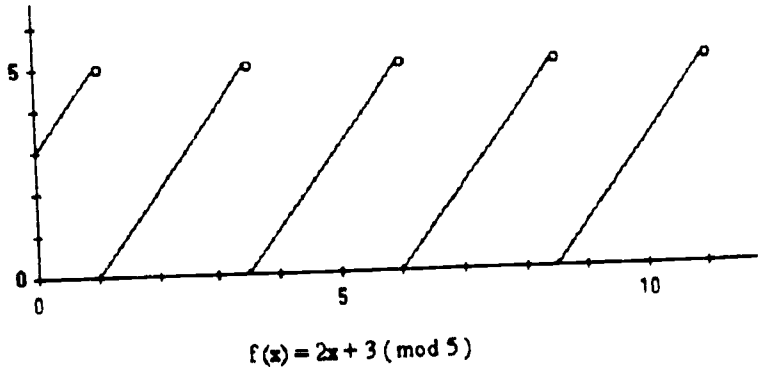


Figure 23. f(x) = 2x + 3(mod 5)

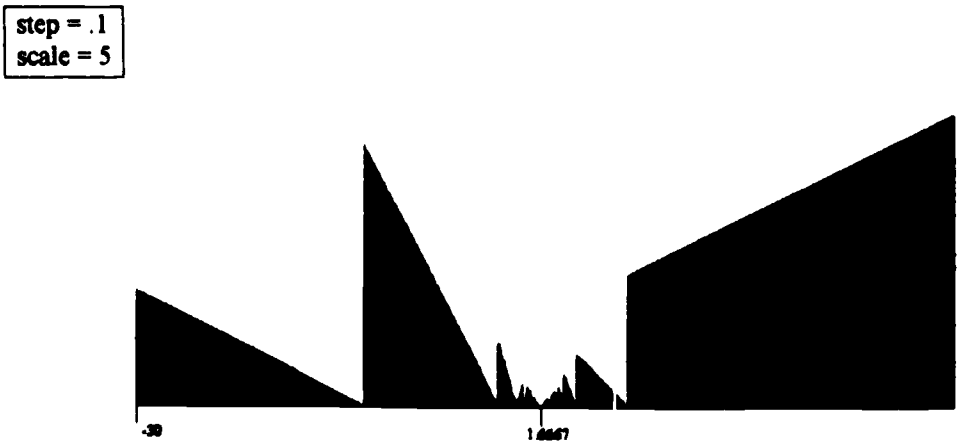


Figure 24. f(x) = 7x + 2 (mod 3x + 5)

What kinds of questions might project-minded high school students ask about this expression and its graph? A number of possibilities are suggested by the size, shape and location of the teeth. For instance, at many points of the graph, the downward sloping edge of the teeth appears to meet the x axis. For what values of x is the value of the expression equal to zero? Does each tooth have a straight line for its downward slope? If so, does each tooth (or pair of teeth) have its own unique downward slope? What about the curve connecting the peaks of all the teeth to the right of 5/3. Are there infinitely many peaks (although very small) as you approach the point 5/3? Are they on a straight line? If so, what is that line? Where is the graph continuous? Where are its discontinuities? These and other questions can be generalized for the expression Ax + B (mod Cx + D). Here is a topic for serious investigation and which might serve as the basis for a science fair project.

Students interested in such questions are certainly not limited to the consideration of expressions involving only linear terms. Figure 25 shows the graph of $x^2 - 40{,}000 \pmod{\sqrt{(x^2 - 20500)}}$. Focusing on the w-shaped object in the center of the graph, one might ask whether the curve in this region is continuous or if there are any other locations where the curve might be continuous. The apparent symmetry of this curve about the origin could also be investigated. At this point, it should be pointed out that the actual complexity of the graph is far more intricate than revealed by the plot shown in Figure 25, which was created by stepping across the number line in regularly spaced intervals of 1 unit per pixel. The details of each small region must be investigated using suitable enlargements of the region. This is accomplished by reducing the step size. For example, Figure 26 shows an enlargement of a portion of the region just to the right of the w-shaped object in the center of the graph in Figure 25.

So far, we have examined the graphs of both linear and quadratic expressions given in terms of one variable, x, by plotting the value of each expression along the y axis. An extension of this approach is to define an expression in terms of two variables, x and y, and to plot the value of the expression along the z axis. In representing such expressions graphically, we will use surface plots (mesh) and color coded raster graphics. The first example of this approach is

$$x^2 + y^2 - 40{,}000 \sqrt{(\text{mod } (x^2 + y^2 - 20500))},$$

an extension of the expression used to plot Figures 25 and 26. A surface (mesh) plot of this expression is shown in Figure 27 and a gray scale raster graphic is shown in Figure 28. Although both of these graphics offer insight into the attributes of the surface defined by the expression, they do not offer the same insights. Figure 27 emphasizes the fractured, erratic nature of the surface. Figure 28 suggests that there are intricate, subtle details to be found in the apprently chaotic landscape of Figure 27. This impression is immediately heightened when the viewer sees the image in full color (see Color Print 1).

In spite of the visual appeal of such graphics, it is not always wise to accept them at face value. In other words, seeing should not always be the same as believing when working with scientific visualizations. For instance, many of the interesting details seen in Figure 28 and Color Print 1 are not "really there" at all. That is, the raster graphic images seen in Figure 28 and Color Print 1 contain both good and bad information about the object being visualized.

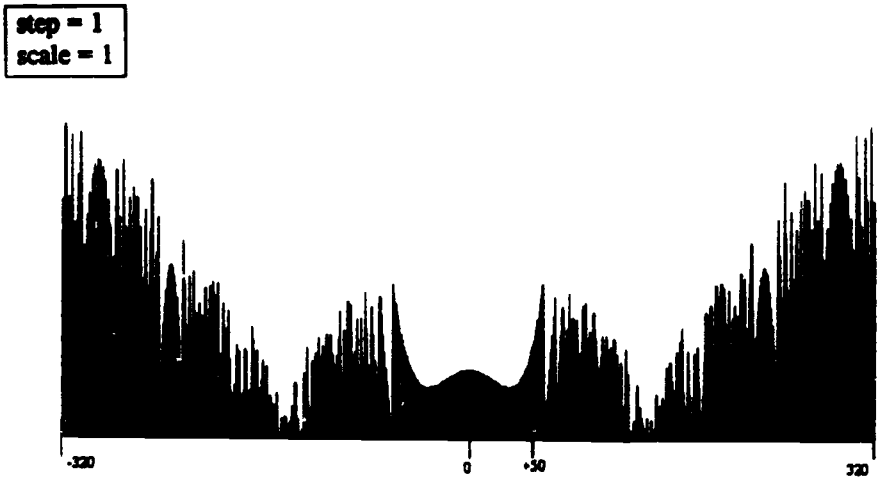


Figure 25. $f(x) = x^2 \cdot 40{,}000 \pmod{\sqrt{(x^2 \cdot 20{,}500)}}$

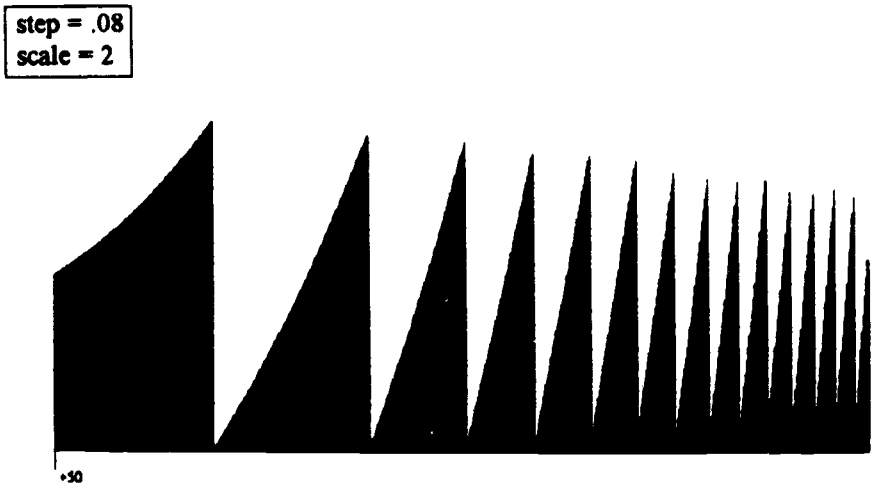


Figure 26. Zoom on Figure 25

As Alex Pang explains in his chapter, every attempt at scientific visualization is ultimately conducted in a computer visualization environment with inherent limitations on data sampling and graphic rendering. In the case of Figure 28 and Color Print 1, the complexity of the object being visualized so exceeds the resolution of any existing computer's sampling and rendering procedures that misleading graphical elements are introduced. The point of this observation is that efforts at visualization must include critical thinking about the relative complexities of the object being visualized and the system doing the visualization.

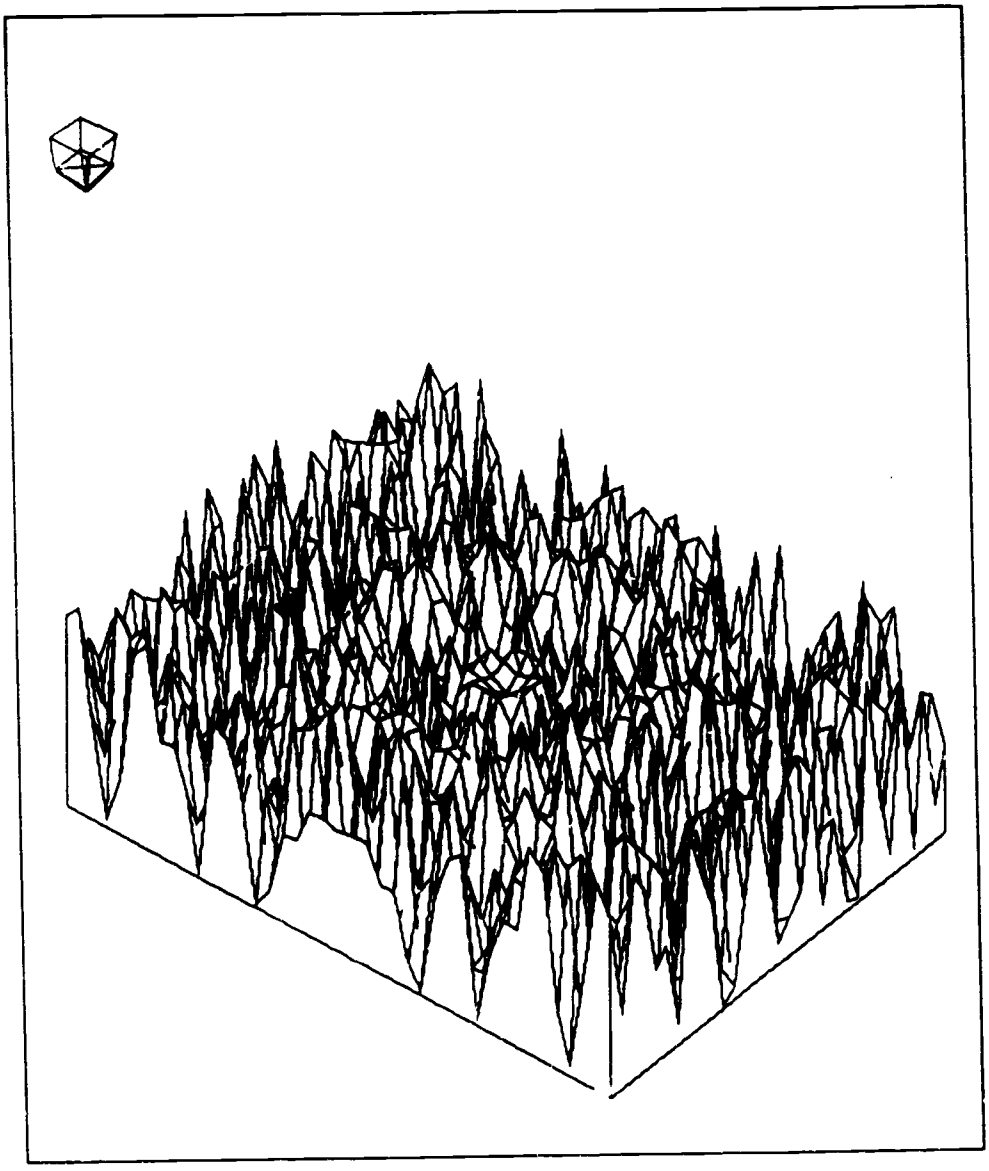


Figure 27. Surface mesh plot

The procedure used to create Color Print 1 was mathematically straightforward but technologically sophisticated. First, a computer program was written in C that computed the value of the expression $x^2 + y^2 \cdot 40{,}000 \pmod{\sqrt{(x^2 + y^2 \cdot 20500)}}$ for a set of 307,200 regularly spaced points in the X-Y plane near the origin (see Program Listing 8).

Figure 28. B&W raster graphic

Program Listing 8

```
#define mnt   9            /* Max Number of Terms */
#define cvt   double       /* Calculation Variable Type */
#define tcvt  (cvt)        /* To Calculation Variable Type */
#define tf    (double)     /* To (double) Float */
#define cta   fabs         /* Calculation Type of Absolute values */

typedef struct                  /* Each term in f(x,y) is of form: */
 {                          /* tk*[(X*ix+xk)^ex]*[(Y*iy+yk)^ey] */
  int nt;                   /* Number of Terms */
  double ex[mnt-1], ey[mnt-1];   /* Exponent on the X, Y */
  cvt ix[mnt-1], iy[mnt-1];      /* Inner constant of X, Y */
  cvt tk[mnt];                  /* Term Konstant */
  cvt xk[mnt-1], yk[mnt-1];      /* X, Y Konstant */
  double rt;                    /* RooT of equation */
 } funct;

typedef struct
 {
  int amx, amy, amc,     /* Absolute Max X, Y, Color */
     gd, gm, cm;         /* Graph Drive, Graph Mode, CMode */
  int mc,                /* Max Color */
```

```
    lipx, lipy,       /* Lower Left Image Position:  X, Y on the screen */
    uipx, uipy;       /* Upper right Image Position:  X, Y on the screen */
 cvt sx, sy, stx, sty,  /* Start X Y, STep X Y */
    gl;              /* Ground Level */
 double sd;           /* Scale Down (or up) */
} params;

/* This is the main drawing procedure.  It does the final
   calculation and puts the value in an array to stored in
   a file or as a color on a graphics screen. */

char dp(fz,fn,p,tom)      /* Draw Picture */
funct *fz, *fn;          /* the two sets of parameters for the functs */
params *p;               /* the parameters for drawing the picture */
int tom;                 /* Type of Mod:  FMOD or RMOD */

{
cvt xc, yc, cc;          /* X, Y, and Color variables for Calculating */
int x, y;               /* X Y posotions on screen */
char c='a';              /* to test what character was typed */

/* SET UP GRAPHICS AND PRINT INFORMATION AT TOP OF GRAPHICS SCREEN */
initgraph(&p->gd,&p->gm,"");
gotoxy(1,1);
printf("T to terminate drawing; \n");
/* */ printf("sx=%lf sy=%lf stx=%lf sty=%lf sd=%lf ",
        p->sx, p->sy, p->stx, p->sty,  p->sd);
 if (tom==1) printf("fmod\n");
     else printf("rmod\n");
 printf("AFTER PICTURE DRAWN: S to save;  then Z to Zoom.\n");

 for(y=p->lipy,  yc=p->sy; y>=p->uipy; y--, yc=yc+p->sty)
   for(x=p->lipx, xc=p->sx; x<=p->uipx; x++, xc=xc+p->stx)
    {
 /*************************************************************************/
     cc=calcf(fn,xc,yc);
      if (cc!=0)  /* here is where the color is assigned to the screen */
     {
      if (tom==1) cc=tcvt(fmod(calcf(fz,xc,yc),cc) - p->gl) * p->sd;
          else cc=tcvt(rmod(calcf(fz,xc,yc),cc) - p->gl) * p->sd;
     }
    putpixel(x,y,(int)(cc));
 /*************************************************************************/

    if (kbhit()!=0)   /* CHECK IF A KEY WAS HIT */
     {
      c=getch();
      if (c=='t') { y=-1; x=p->amx+1; }
          else c='a';
     }                /* end of IF KBHIT()!=0 */
    }                 /* end if FOR(X=0 to P->MX) */
```

```
return(c);
}
/* This procedure calculates the value of a function based on the
   values of the parameters in F and the X Y values and returns
   this value through the function. */

cvt calcf(f,x,y)          /* CALCulate Function */
funct *f;                 /* calculates the function, fz or fn, */
cvt x, y;                 /* depending on which is passed to it */

{
 int c;                   /* a counter */
 cvt t=f->tk[f->nt-1], xt, yt;  /* Total of all the terms added up */

 if (f->nt>1)
   for(c=0; c<f->nt-1; c++)
    {
     xt=power((f->ix[c]*x+f->xk[c]),f->ex[c]);
     yt=power((f->iy[c]*y+f->yk[c]),f->ey[c]);
     t=t+xt*yt*f->tk[c];
    }
 if ((t!=0) && (f->rt!=v1))  t=power(t,f->rt);
 return(t);        /* returns always positive values because of log */
}

/* The functions calculates the value of a number N raised to the
   P power.  The main purpose of this function is to preserve the
   sign of the number N.  For example, power(-3,2)=9, but
   power(-3,3.5)=-46.76537...  */

cvt power(n,p)            /* This calculates N to the P */
cvt n;
double p;

{
 int pt=(int)(p);         /* integer part of the exponent, PowerTemp */
 cvt nt=0;                /* NumberTemp, contains finally N to the P */

 if ((n!=0) && (p!=0))
  {
   nt=v1;
   p=p-tf(pt);

   if (pt<0)  while (pt<0) { nt=nt/n; pt++; }
      else  while (pt>0) { nt=nt*n; pt--; }
   nt=nt*exp(log(cta(n))*p);
  }
   else if (p==0) nt=1;
```

```
return(nt);
}
/* This is my own version of the MOD operator with the following
  differences:
                        fmod     rmod
   7.3000 (mod  4.5000) =   2.8000   2.8000
   7.3000 (mod -4.5000) =   2.8000  -1.7000
  -7.3000 (mod  4.5000) =  -2.8000   1.7000
  -7.3000 (mod -4.5000) =  -2.8000  -2.8000  */

cvt rmod(n/m)                    /* Real MOD, same as fmod(double x,y); */
cvt n, m;

{
long q=(Long)(n/m);              /* Quotient */
cvt rm=n-m*tcvt(q);               /*ReMainder */

if (rm*m<0.0) rm=rm + m;
return(rm);
}
```

Program Listing 8 shows a portion of the code used to calculate the data that is the basis of the visualization shown in Color Print #1, the most important function being DP (Draw Picture). In DP, the value of the mod expression is calculated point by point and plotted on the screen or put into an array to be stored into a file. The other three functions are support procedures and are described in comments found in the following program listing. It should be noted that the algorithms presented here are written to work with a Borland C compiler. The algorithms are not necessarily efficient or easy to read nor are they complete and ready to compile. They are presented here to illustrate the approach taken in writing the program. Complete copies of the source and executable code for both the PC and Cray versions of this program are available on disk (specify format) from the authors for $5.00.

The complete version of this program was sent via the Internet to a Cray Y-MP supercomputer at the National Center for Supercomputing Applications (NCSA) at the University if Illinois at Urbana-Champaign for compiling and execution. The 480 X 640 array of floating-point numbers corresponding to the expression's value at each of the 307,200 poir s in question was then transfered back to Montana State University over the Internet and saved on a Macintosh IIci computer. Visualization of the data was accomplished using the NCSA Scientific Visualization Software Suite and PDImage, both of which are in the public domain. Both of these tools are easy to use and offer a wide variety of data analysis and display options. For example, in Figure 29 PDImage is used to plot the functional values along a horizontal line segment drawn through the origin. This feature is particularly helpful when investigating complex surfaces, as it allows the researcher to first simplify the problem to that of examining a collection of cross sections of the surface.

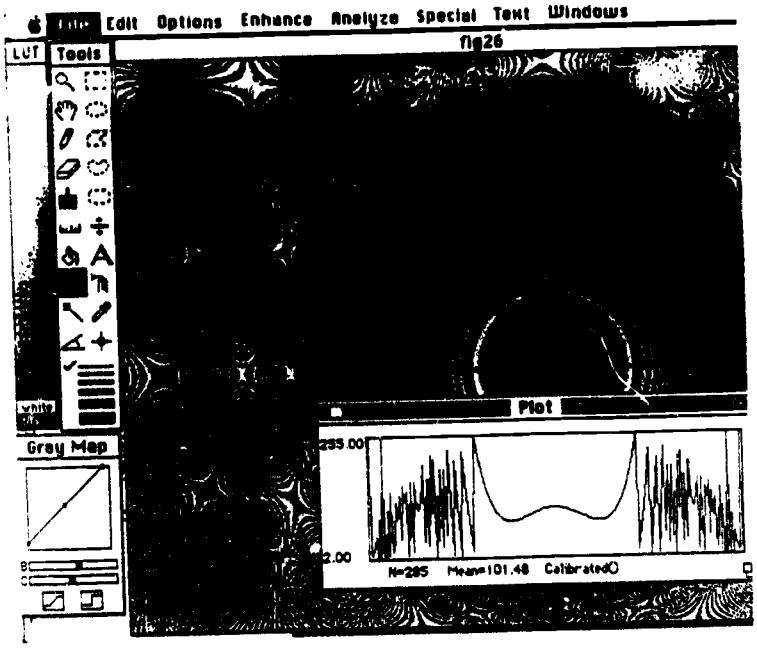


Figure 29. Cross section using PDImage

By examining visualizations such as those shown in Figures 28, 29, and Color Print #1, questions similar to those posed with regard to Figure 24 and 25 may be asked. For example, how do cross sectional slices (see Figure 29) change as the line defining the slice moves away from the center? Are there regions where the surface defined by the function is both continuous and differentiable? What shapes do the 3-dimensional "teeth" take on? And so on. Posing such questions is itself a creative mathematical endeavor rarely encountered in high school or undergraduate mathematics. Visualization tools make this type of activity possible for novice mathematicians who might otherwise never attempt a mathematical investigation on their own. In that sense, visualization tools open new windows on mathematics to students.

Looking Ahead

The use of images to stimulate curiosity, define relationships, and express emotion is one of mankind's most characteristic traits. Using a variety of technologies, businessmen, governmental officials, and a host of other professionals routinely create, transmit, and receive images from all over the world. As fiber optics cables reach more and more communities, schools will gain access to the international academic telecommunications networks like Internet and Bitnet. When that happens, the teachers and students in those schools will join a kind of international community in which the exchange of ideas takes place at the speed of light. Powerful computing resources will become available. Vast storehouses of information will evolve. And the experience we call "going to school" will gradually change as teachers and students adapt to life in the Information Age.

We look forward to a variety of changes in school science and mathematics. As the cost of computer resources comes down, we believe that students will spend more and more time organizing, analyzing, and visualizing meaningful data. As patterns are found in the data, abstractions will be introduced to summarize the students' findings. And as school mathematics becomes more inductive, it will be used more and more as a tool of school science. In science classes, computer modeling and simulation will help students to understand the invisible world of the atom, unimaginable forces near the surface of a black hole, and the complex relationships found in all living systems. Even simulation itself will come under study as students ask the question "To what extent does a given simulation adequately model the real world?" And with greater and greater frequency, the exclamation, "Now I see!" will ring throughout classrooms around the world.

References

Barnsley, M. (1988). *Fractals everywhere*. San Diego: Academic Press.

Chaos: The Software. Computer software available from Autodesk, 2320 Marinship Way, Sausalito, CA 94965.

Desktop fractal design system. Computer software available from Academic Press, Inc., 1250 Sixth Ave, San Diego, CA 92101.

Gleick, J. (1987). *Chaos: Making a new science*. New York: Viking.

LogoWriter. Computer software available from Logo Computer Systems, Inc., New York.

Mandelbrot, B. (1977). *Fractals: Form, chance, and dimension*. San Francisco: W.H. Freeman and Co.

Mandelbrot, B. (1982). *The fractal geometry of nature*. New York: W.H. Freeman and Co.

McCormick, B.H., DeFantini, T.A., & Brown, M.D. (1987). Visualization in scientific computing. *Computer Graphics, 21*(6).

NCSA scientific visualization software suite. Computer software available from the National Center for Supercomputing Applications, 605 East Springfield Avenue, Champaign, IL 61820-5518.

PD image. Computer software available by anonymous FTP from alw.nih.gov in the /pub/image directory. Author: Wayne Rasband, wayne@helix.hih.gov.

Peitgen, H.O., & Richter, P. (1986). *The beauty of fractals*. New York: Springer-Verlag.

Peitgen, H.O., & Saupe, D. (Eds.) (1988). *The science of fractal images*. New York: Springer-Verlag.

Thomas, D.A. (1989). Investigating fractal geometry using LOGO. *Journal of Computers in Mathematics and Science Teaching, 8*(3).

Working groups of the commission on standards for school mathematics. (1989). *Curriculum and evaluation standards for school mathematics*. Reston, VA: NCTM.

Zimmerman, W., & Cunningham, S. (1991). Editor's Introduction: What is mathematical visualization? In W. Zimmerman & S. Cunningham (Eds.), *Visualization in teaching and learning mathematics*. Washington, DC: Mathematical Association of America.

Chapter 6

Visualizing Computer Science

ROCKFORD J. ROSS

Introduction

If an ordinary picture is worth a thousand words, what is the value of a full-color, three-dimensional, animated view of a complex scientific or mathematical process whose parameters can be tinkered with on the fly? This question has captivated the imaginations of science and mathematics educators, as powerful computers capable of producing high-quality graphical images become more readily available in the classroom. Chemistry, mathematics, physics, and virtually all other sciences come alive in the context of well-done computer visualizations, clarifying concepts that may otherwise be difficult or time consuming to teach and learn. Teachers find that computer visualization aids allow them to explore issues in the classroom that before seemed too dry and murky for most students to grasp. And the students, themselves, are frequently enamored by the immediate feedback and element of fun afforded by the computer, often spending time on their own to experiment with scientific concepts through the visualization system. The potential benefits of computer visualization on science and mathematics education are simply enormous.

In this chapter we explore the use of computer visualization as an aid for teaching and learning computer science. At first it may seem surprising—or perhaps even a little odd—that computer visualization techniques would be applied to computer science. After all, what we see on the computer screen (even the visualization of, say, a physics experiment) is actually a visualization of computer science as well, isn't it? Well, no, not really. To understand why, and to see how computer visualization can play an effective role in teaching and learning computer science, a clarification of this discipline is probably in order. The following section, therefore, provides an introduction to computer science as a prelude to the later discussions of computer science visualization systems;

no previous knowledge of computer science on the part of the reader is assumed in this section. The third section then begins a discussion of visualization techniques that work for computer science, beginning with program animation. Algorithm animation is the topic of the fourth section. The fifth section describes other visualization techniques of interest in computer science education, and the final section gives a summary of the chapter.

Background

While most young educators today are trained in the use of computers and understand something about their applications, fewer have actually had training in computer science. In this section a brief tutorial on the science of computing is provided as a basis for the later sections that cover computer science visualization systems. Readers who already have a good grasp of computer science can skip to the third section directly.

Computer Science—A Definition

Computer science as a discipline always seems to be embroiled in some identity crisis or another: just what is computer science anyway? Is it programming in Basic, Logo, or Pascal? Is it the application of the computer to common problems, such as word processing, databases or spreadsheets? Is it the design of computer hardware? Actually, none of these things captures the essence of computer science, but it isn't any wonder that there is a cloud of confusion surrounding the discipline: unlike other sciences, such as physics or chemistry, computer science is associated with a tangible, real-life object—the computer—that nearly every person interacts with in one way or another. Therefore, nearly everyone has some (possibly erroneous) idea about what computer science must be. And therein lies the rub. Computer science is really not about the compute*r* at all, but about comput*ing*. (Why don't we call the discipline *computing science* rather than *computer science*, then? Good question. As noted already, the discipline has faced numerous identity crises, one of which was what to name it. Although other names may be better, computer science has stuck, for better or for worse.)

While many different formal definitions of computer science have been proposed, most can be distilled simply to this: *Computer Science is the study of problems and their solutions on a computer*. An elaboration of this definition and a thorough description of the entire academic field of computing can be found in the widely referenced article *Computing as a Discipline* (Denning et al., 1989); this paper was the inspiration for the most recent revisions to computer science curricula at post-secondary institutions, as described in *Computing Curricula 1991* (Tucker et al., 1991). It is also being used as a basis for deciding what should be taught for computer science in secondary schools (Merritt, 1992; Taylor et al., 1992). The trend is towards incorporating more computer science in secondary schools rather than just computer programming or computer literacy; as these changes gradually occur, the need for supporting teaching and learning aids, such as the visualization systems described here, will become more important.

Our shortened definition of computer science has three focal points: *problems*, *solutions*, and *computers*. Any computer visualization of computer science should therefore shed light on these concepts. To better understand them, let's examine them in order.

Problems

To qualify for study within the science of computing, a problem must pass muster on a few criteria:

1. A problem must, in general, have an infinite number of instances. If this were not the case, the answers to all of the finite number of instances of a problem could be computed just once and stored in a table for future reference. This approach obviously won't work, however, for problems that have an infinite (or impractically large) number of instances. In this case rather than computing answers ahead of time and providing a table of the answers as the solution to a problem, one must instead provide a *method for computing* an answer for any problem instance as the need arises. Then, when the answer to a particular instance of the problem is required, the method for computing the answer is applied to the problem instance to get the answer.
2. The problem must be well-defined. If we couldn't agree about what the problem is, we surely wouldn't agree on a solution to the problem. Thus, while something like "What is the meaning of life?" is a problem for most people, it does not lie within the purview of computer science because it is not well-defined.

Example 1

An example of a simple problem that meets the above criteria is: Compute and print the sum of the first n nonnegative integers. This problem qualifies because it has an infinite number of instances, one for each nonnegative integer n, and it is well-defined. In other words, one could not build a table in which the sum of the first n integers could simply be looked up for each n, so instead a method for computing these values as needed must be provided. Also, everyone would agree that a solution to this problem is any procedure that comes up with the sum of the first n integers, so there is no question about the meaning of this problem. ■

Solutions

In computer science, a solution to a problem is, as noted above, a method that will compute the correct answer for any instance of that problem. We call such a solution method an *algorithm*. To qualify as an algorithm, a proposed solution must:

1. be composed of simple steps that can be carried out "mechanically." (A program written in a standard programming language satisfies this criteria.)
2. compute the correct answer for every instance of the problem. (This implies that the algorithm cannot go on forever, but must halt after a finite number of steps with the correct answer for each problem instance.)

Example 2

An algorithm for the problem of example 1 can be written in general terms, such as: "Obtain a value for n. Sum each of the integers between 0 and n inclusive. Report the total as the answer for this instance." Applying this algorithm to the problem instance $n = 4$ yields the answer 10. For the problem instance $n = 7$, the answer is 28, and so forth. ■

Often, algorithms are presented in a "pseudo programming language" rather than English, because such a presentation is much more precise and closer in form to a computer program.

Example 3

A pseudo programming language algorithm (with comments) corresponding to the general algorithm of example 2 can be written as:

```
input n                          obtain a value for variable n
total ← 0                        assign 0 to variable total
i ←1                             assign 1 to variable i
loop while i ≤ n                 loop while i's value is less than
                                     or equal to n's value
    total ← total + i            add i to total
    i ← i + 1                    add 1 to i
    endloop                      end of loop body
output total                     report the value in total
```

This algorithm will successfully compute the sum of the first n integers for any n greater than or equal to zero. ■

A notion allied to that of algorithm is program. A *program* is the implementation of an algorithm in some real programming language, ready to be run on a computer.

Example 4

A Pascal program that implements the algorithm of example 3 is given below:

```
program sumfirstn (input, output);
    {this program computes the sum of the first n nonnegative
    integers}
 var
    n, i, total: integer;

 begin
    {input a value for n}
    writeln('Please type a nonnegative integer for n.');
    read(n);
```

```
{keep inputting until a nonnegative integer is typed}
while n < 0 do begin
    writeln('You entered a negative value for n.');
    writeln('Please enter a nonnegative integer.');
    read(n)
    end;

{compute and print the sum of the first n integers}
total := 0;
i := 1;
while i <= n do begin
    total := total + i;
    i := i + 1
    end; {loop}
writeln('Answer:');
writeln('The sum of the first ', n:1, ' integers is: ',
        total:1)
end. ■
```

In common discourse the words "algorithm" and "program" are often used interchangeably with no harm. As the previous two examples illustrate, however, there are differences between a pseudo programming language algorithm and an implementation of that algorithm as a program. The algorithm captures the idea of the method to be used in the solution of a problem, whereas a program focuses on the specific details of getting that solution to run on a computer. An algorithm also does not need to include checks for invalid data, because as a mathematical object it is only defined over valid instances (e.g., nonnegative integers) and is simply undefined elsewhere. In a real program, however, such checks should be made for robustness, because a user can type in invalid values which the program will attempt to evaluate. Finally, a good program should include input and output statements that make it user-friendly, whereas this is unnecessary in an algorithm.

Computers

A computer is simply a device that can carry out a program automatically. Although there are many different types of computers available, their sole purpose is to "mechanically" carry out the individual steps of a program in order to produce answers to instances of the problem supplied. A computer is constructed with only a few hardware instructions (usually called the computer's *machine language*) that it can perform, such as add, subtract, multiply, divide, compare, and jump. In the final analysis, programs intended to run on a particular computer must be presented in terms of these simple instructions that are designed into the computer. (Programs called *compilers* are available for translating standard programming languages, such as Pascal, into a computer's machine language.)

Because a computer is a physical device, it obviously has limitations. For example, the algorithm given in example 3 to compute the sum of the first n nonnegative integers presents a solution that will work for any n greater than or equal to 0. A computer that executes the Pascal implementation of this algorithm given in example 4, however, will

only work for input values and computed values (i.e., *sum*) within the integer size limitations of the computer.

The Science of Computing

The previous sections on problems, their solutions as algorithms and programs, and computers that can execute programs provide a basis for understanding the kinds of questions explored in the science of computing. These include the following:

Limitations of Algorithms. As described earlier, algorithms must eventually be expressed as programs that can run on a computer. Thus, the individual statements that can be used in constructing an algorithm are ultimately dependent on the relatively small number of instructions that a computer can carry out automatically. Given this limitation, are there well-defined problems that have no solution (i.e., no algorithm)? The answer is yes. There are many (an infinite number, actually) interesting problems that have no solution. One well-known one is the halting problem. This problem can be presented in this fashion:

> A computing facility at a university has noticed that lots of time and computer resources are wasted by students in the Pascal programming course, because as novice programmers, these students often write programs that go into infinite loops that waste computer time and other resources. Would it be possible to write a program—call it HaltTester—that screens student programs for infinite loops before the programs are actually run on the computer (i.e., determine whether these programs will halt or not)? An instance of this problem is an arbitrary Pascal program and the data that this program will be run on this time. Clearly, there are an infinite number of instances of this problem, one for each conceivable Pascal program and data for that program. Also, the problem itself is well defined: an answer for a given instance (Pascal program and its data) is, "Halts!" if that Pascal program would halt when executed on the given data, and "Won't halt!" otherwise.

It can be proven that program HaltTester cannot be written. That is, there is no solution (algorithm) for this problem. Thus, one of the objectives of computer science is to identify problems that are unsolvable, so that time will not be wasted trying to develop algorithms for them.

Limitations of Computers. It may appear from the previous section that the limitation on the power of algorithms to solve problems may lie with the computers themselves. After all, the statements that can be used for the individual steps in an algorithm are ultimately limited by the instruction set of the target computer. Perhaps it would be possible to design more powerful computers with richer instruction sets that allow algorithms to be developed that solve problems that cannot be solved with current technology. Unfortunately, this also appears unlikely. Just as multiplication can be expressed in terms of addition, so can all of the other usual operations we can think of be composed from just a small number of elementary operations. In fact, a cornerstone of computer science theory is the Church-Turing Thesis, which essentially states that if a problem is unsolvable with algorithms and computers as we currently understand them, it is unsolvable, period. Students of computer science need to be cognizant of this important claim.

Classifying Algorithms. Having acknowledged the fact that there are unsolvable problems, computer scientists focus their attention on those that are solvable. For each solvable problem, there are an infinite number of algorithms that solve the problem. Is there a measure that can be applied to determine which of the algorithms is best? One common measure is the amount of time required to execute an algorithm as a function of the "size" of the problem instances input to the algorithm.

Example 5

Consider again the problem of computing the sum of the first n nonnegative integers. The algorithm for this problem as given in example 3 is repeated below (this time without comments). Here the statements are numbered for reference.

```
1  input n
2  total ← 0
3  i ← 1
4  loop while i ≤ n
5      total ← total + i
6      i ← i + 1
7      endloop
8  output total
```

An instance of this problem is a specific value for n. If we count the number of statements that will need to be executed in arriving at an answer for any value input for n, we see that statements 1-3 will each be executed once, as will statement 8. Inside the loop, statements 5-7 will be executed n times, depending on the value of n. Similarly, the loop statement itself (statement 4) will be executed n times, plus one more time when it is determined that $i > n$, at which point the loop is exited. Thus, $4n + 5$ statements will need to be executed for any input value of n in order to arrive at an answer: if $n = 3$, 17 statements will be executed in computing the sum of the first n integers; if $n = 1000$, 4005 statements will be executed in computing the sum of the first n integers; and so forth. ■

The formula $4n + 5$ in the above example is a measure of how complex the algorithm is with respect to how much time is required to process the algorithm with different values of n. We thus refer to the formula as a *time complexity* measure of the algorithm. In this case we say that the given algorithm has *linear* time complexity, because the formula of the number of statements that will be executed is a linear function of n, the input value for the algorithm.

Example 6

A different algorithm that solves the same problem is given below. Here, the sum of the first n integers is computed by starting with n in *total* and then adding $n - 1$ to *total*, $n - 2$ to *total*, and so on.

```
1   input n
2   total ← n
3   i ← n - 1
4   loop while i > 0
5       total ← total + i
6       i ← i - 1
7       endloop
8   output total
```

In this case, since *total* is initialized to *n* to start with, this eliminates one of the values that needs to be added to *total* in the loop. A statement count analysis thus yields the formula 4n + 1 for this algorithm (the four statements of the loop will be executed one less time than in the algorithm of the previous example). ■

In comparing the algorithms of examples 5 and 6 we would have to say that the one in example 6 is better in terms of its time complexity. However, both have linear time complexity, and the differences between the two are so minuscule as to hardly consider. For example, if $n = 1000$, the algorithm of example 5 will require that 4005 statements be executed, whereas 4001 will be executed in the algorithm of example 6. The differences pale as *n* gets larger, and, in fact, would be virtually undetectable if the algorithms were implemented and run on a computer.

What *would* be of interest, however, is an algorithm that solves the same problem but has a time complexity formula that is less than linear.

Example 7

If someone were to attack the problem of computing the sum of the first *n* nonnegative integers who had studied enough math, he or she might remember that there is a closed form formula for this value:

$$\text{sum of first } n \text{ integers} = \frac{n(n+1)}{2}$$

Thus, another algorithm for this problem is

```
1   input n
2   total ← (n * (n + 1)) / 2
3   output total
```

The time complexity formula for this algorithm is just 3, regardless of the size of the value input for *n*. ■

The algorithm of example 7 has a time complexity formula that is constant, so we say that the time complexity of that algorithm is *constant*. It is obviously much better than the previous two algorithms covered.

It should be clear from the last three examples that the search for a "best" algorithm (in terms of time complexity) for a given problem is an important focus of computer

science. Students learning this kind of analysis can be helped by computer visualization systems that clearly illustrate comparative time complexities of various algorithms for a problem. Examples of such systems will be covered later.

Inherent Problem Difficulty. A closely related focus of computer science is the classification of problems (rather than algorithms) according to their difficulty. For example, if we could prove that the best algorithm for a particular problem has linear time complexity, then we would say that the *problem itself* has linear time complexity. The problem of computing the sum of the first n nonnegative integers thus has constant time complexity, because the best algorithms for solving this problem have constant time complexity (that's the best possible time complexity for a problem).

Are there problems that are "more difficult" (have higher time complexities) than others? A little reflection tells us that this must surely be the case.

Example 8

We've all had experience looking up words in a dictionary, and we know that we don't need to look at every word in the dictionary to find the one we're looking for. So, although we could design an algorithm to solve this problem that *did* examine each word in the dictionary (with linear time complexity over the number of words in the dictionary), we know there must be a cleverer way. One way would be to open the dictionary in the middle and check whether the desired word is in the first half or the second half of the dictionary. We take whichever half the word is in and repeat this process until the word is found. This leads to a $\log_2 n$ time complexity—much better than linear. On the other hand, we also know that as the dictionary gets larger, we somehow are going to need to do more work. So we are unlikely to find a constant time algorithm that requires the same time to look up words in a dictionary regardless of the size of the dictionary.

Now consider the problem of sorting a list of words into alphabetic order. Most teachers have had this experience on a relatively small scale trying to rearrange tests into alphabetic order by the last names of the students. Again, it is intuitively clear that, unlike the problem of looking up words in a dictionary, this problem cannot be accomplished without looking at each word. Also, it seems pretty clear that we can't get by with just looking at each word once, because we will need to make comparisons between words somehow. Indeed, the best algorithms for this problem can be shown to have $n*\log_2 n$ time complexity, where n is the number of words to be rearranged. (Readers interested in reading more about algorithms that have this time complexity can find them in the chapter about sorting in any standard textbook on algorithms and data structures, such as Weiss, 1992. For example, the mergesort and quicksort algorithms have this time complexity.) ■

As this example illustrates, problems themselves range in difficulty, and it is important for the student of computer science to be able to analyze—or at least recognize—the different time complexities of problems. Computer visualizations that compare algorithms can underscore this fact, as described later in this chapter.

Computer Visualization and Computer Science

As this section has demonstrated, the three areas of focus for a computer scientist are problems, solutions (algorithms and programs), and computers. Actually, however, problems are understood in the science of computing only through their algorithms (or lack thereof), and computers are generally only of interest in that they are available for carrying out the programs that implement the algorithms. So, in the final analysis, algorithms are the overriding and primary concern of computer scientists. This has led some scientists to name the core study of computer science *algorithmics*, thus implying that the computer scientist is an *algorithmician*. A highly recommended book that describes this core area of computer science in layman's terms is *Algorithmics: The Spirit of Computing* (Harel, 1992).

Thus, any computer visualization system intended to elucidate the field of computer science should focus on algorithmics, particularly illustrating how algorithms and programs work and providing insight into the fundamental differences among algorithms as well as their time complexities. As described in the next sections, successful visualization systems exist for these purposes.

Program Animation

Can computer visualization systems be used to teach and learn computer science? Are there any such systems available? Do they work? The answer to the first and last of these questions is a hearty "yes!" Availability is still an issue, but progress is being made.

We begin in this section by describing the visualization systems we call program animators. As discussed in the previous section, a program written in some programming language (e.g., Pascal) can be thought of as the implementation of an algorithm. Computer science students must become intimately familiar with programs and programming in order to master the science. Program animators can greatly enhance this process.

The Problem: Teaching and Learning Program Execution Dynamics

As those of us who have taught computer science have discovered, one primary impediment to a clear grasp of programming and computer science is a lack of understanding on the part of students new to computer science concerning the relationship between a *program* and its *execution*. Program animation systems offer a computer-based, visual remedy for this problem.

Consider again the Pascal program *sumfirstn* of example 4. That program—in its static, finite form as presented in the example—is a solution to the problem of computing the sum of the first n nonnegative integers. However, when the program is actually applied to some explicit instance of the problem (say $n = 5$), it must be run on a computer. At that point the program is no longer static, but dynamic. Program statements are executed in order, variable values change, tests on variables are made that determine how many times the loop will be iterated, and so forth. Many students have difficulty making the connection between a (static) program and how that program behaves as it is executed. Consequently, instructors are faced with the task of explaining this

ERIC

correlation. It is safe to say that a student who never grasps program dynamics will in turn never understand how programs solve problems and as a result never become an effective programmer.

Just how does an instructor explain how a program works as it is executed? Those who have tried find this to be a somewhat daunting experience. Traditional approaches have the instructor first developing a program at the blackboard and then "playing computer" on the program. Usually this proceeds as follows: The instructor represents computer memory by writing down the names of the variables next to the program on the board. Then, to illustrate program execution, he or she draws an arrow next to the current line of the program being "executed" and discusses what will happen when the computer actually executes that line. This may cause a variable value to change, so the instructor erases the old value of the variable and replaces it with its new value. Then the arrow pointing to the current line of execution is erased and redrawn at the next line to be executed, and the process repeats.

Of course, this entire process is highly prone to human error, and recovering from an error in front of a class of already bewildered students is nearly impossible, because this requires redrawing the illustrations on the blackboard to represent some previous state of program execution (which even the instructor may not be able to recall) and proceeding from there. Even when the presentation progresses smoothly, answering student questions, such as "I didn't catch those last few steps, could you please do them again?" is just as difficult for the same reason. Finally, any student attempting to record this presentation will go home with a hopeless jumble of crossed out values and scattered notes that is virtually uninteiligible: it is nearly impossible to capture the dynamics of a program in execution in static textbook or note form.

The Solution: Computer Driven Program Animation Systems

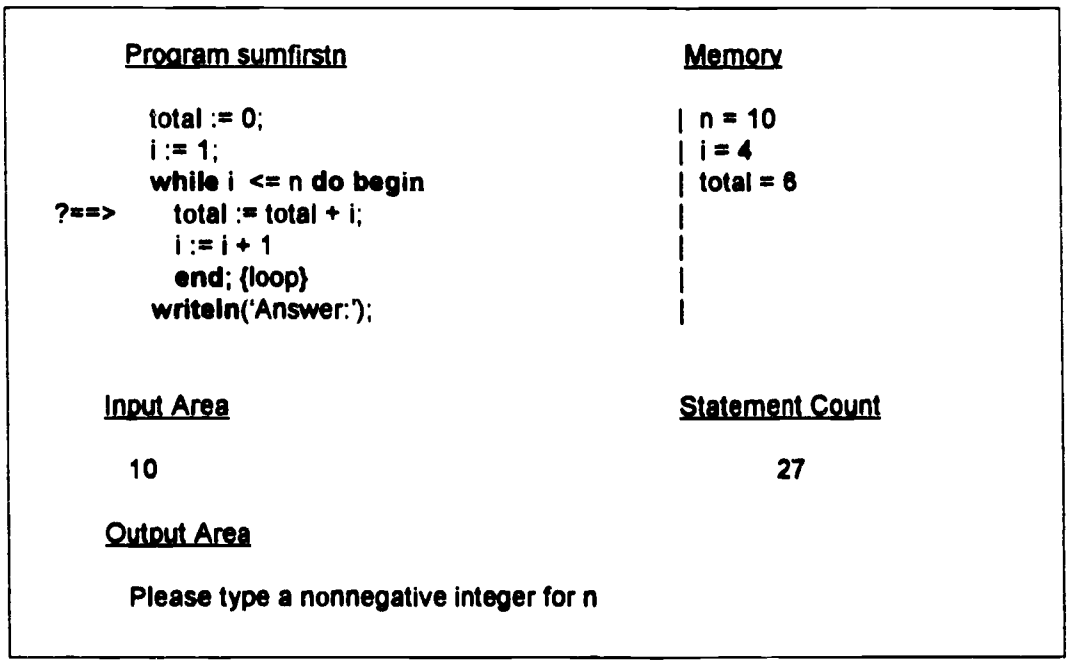
Just think of the doors that would open if the task of explaining program execution dynamics could be automated! Among the things one can envision are these:

1. An instructor could display the animation to an entire class on a wall screen using an inexpensive overhead projection display device connected to a computer, thus eliminating the need for hand animations at the blackboard.
2. Animations would be free of the errors inherent in a blackboard presentation of program execution dynamics.
3. Questions about the last few steps of an animation could be answered simply by backing the animator up and repeating those steps as often as necessary without error.
4. Students could concentrate on the animation without taking notes, because they could be given the same animated program to study on a computer at their leisure.
5. In fact, an entire library of animated programs could be provided on a disk with the animation system so that students could explore any number of programs on their own.
6. Stretching imagination even further, a complete textbook or programming lab manual could be available on a disk (for example, in hypercard form) in which textual discussions could accompany animated programs.

The ramifications of such an animation system are difficult to estimate. Instructors would be able to present many more examples in class, students could perform repeatable experiments on assigned programs, the visual animations would encourage students to explore programs on their own, and the list goes on.

Example 9

During an animation of program *sumfirstn* of example 4, the computer video screen might be organized as shown below (with the use of high resolution graphics and windows, the actual display would generally be much more elaborate).



In this example, which portrays a snapshot of the program after execution has been in progress for some time, the animation screen is organized into five sections. In the program section, just a portion of the program where execution is currently occurring is displayed (notice that only a small, relevant section of the program will fit here at any one time; this portion will be scrolled as necessary). The current line to be executed as the animation progresses is denoted by the ?==> arrow. In the memory section all of the program variables are listed with their current values; at the beginning of program execution, these variables would have no values, which would be indicated by printing a ? symbol in place of the values. The input area is where the user can type values that are needed when input is requested by the program. In this example, the user has typed a 10 in response to an input (**read**) statement executed earlier in the program, indicating that the program should compute the sum of the first 10 nonnegative integers. Notice that this value (10) has been stored in variable *n*, as shown in the memory section. The output area lists the messages that are printed by the output (**writeln**) statements of the program. In the example, execution of an earlier **writeln**

statement has caused the message "Please type a nonnegative integer for n" to appear.

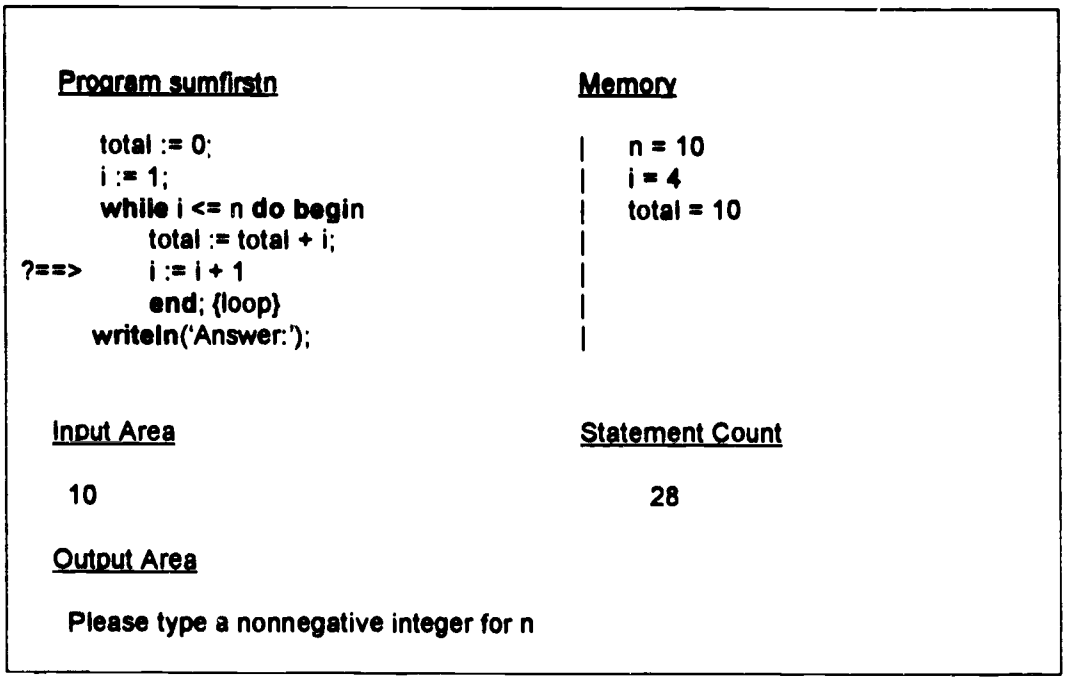
One can envision an instructor presenting this example to a class by projecting this image on a wall screen. The animation has progressed to the point indicated by the ?==> arrow. The current values of the variables are $n = 10$, $i = 4$, and *total* = 6, meaning that the loop has been executed for $i = 1, 2$, and 3, and is now in the 4th iteration. Thus, *total* contains 0 + 1 + 2 + 3, or 6, at present. At this point the instructor can ask the class, "What will happen when this statement is executed (the one pointed to by the ?==> arrow)?" The presence of the question mark on the arrow signifies that it is time to ask this question. The students will hopefully be able to recognize that as a result of executing this statement,

```
?==>    total := total + i;
```

the memory value for *total* will change to 10—the result of adding *total's* value (6) to *i's* value (4). After sufficient time has elapsed to give the students a chance to guess what would happen, the instructor can push the *enter* key on the computer's keyboard to cause the animator to execute the statement in question. The result will be that the question mark will be removed from the arrow, signifying that the statement has been executed, *total's* value will change to 10 in the memory section, and 1 will be added to the statement count, changing it to 28. Thus, the screen will now look like:

```
Program sumfirstn                          Memory

    total := 0;                            | n      = 10
    i := 1;                                | i      = 4
    while i <= n do begin                  | total  = 10
==>     total := total + i;                |
        i := i + 1                         |
        end; {loop}                        |
    writeln('Answer:');                    |


Input Area                                 Statement Count

 10                                                28

Output Area

  Please type a nonnegative integer for n
```

At this point the instructor can ask the question, "Which statement should be executed next?" In this case the answer is simple enough: the statement immediately following the one just executed (pointed to now by the ==> arrow) should be executed. Pressing the *enter* key once more will cause the animator to move the arrow to that statement, again with the ? symbol displayed, yielding:



Except for moving the arrow, the animator changes nothing on the display, allowing the instructor to repeat the process described earlier. ■

The purpose for having the arrow displayed twice at the same statement, once as ?==> before execution and then as ==> after execution is best explained by another example.

Example 10

Suppose that the animation were at the point illustrated below, where *i* now has value 11 and *total* = 55 (just the program and memory portions of the screen are shown):

```
      total := 0;                    | n = 10
      i := 1;                        | i = 11
?==>  while i <= n do begin          | total = 55
          total := total + i;        |
          i := i + 1                 |
          end; {loop}                |
      writeln('Answer:');            |
```

This time when the instructor asks what will happen when the marked statement is executed, students should recognize that *i* is not less than or equal to *n*. When the ***enter*** key is pressed, this is indicated by the animator by replacing the ? with an F (for FALSE) on the arrow; that is, i <= n is now false:

```
      total := 0;                    | n = 10
      i := 1;                        | i = 11
F==>  while i <= n do begin          | total = 55
        total := total + i;          |
        i := i + 1                   |
        end; {loop}                  |
    writeln('Answer:');              |
```

Now, when the instructor asks where execution should continue, the students should recognize that since the loop test is false, the next statement to be executed should be the one immediately following the loop. The animator will indicate this when the *enter* key is pressed once more:

```
      total := 0;                    | n = 10
      i := 1;                        | i = 11
      while i <= n do begin          | total = 55
        total := total + i;          |
        i := i + 1                   |
        end; {loop}                  |
?==> writeln('Answer:');
```

■

Although it is extremely difficult to portray the effects of program dynamics in a book like this, the educational benefits of a program animation system should be readily apparent from this example. Many other features, such as the ability to reverse the animation in response to student questions, various possibilities for skipping over statements that are not of interest, allowing the animation to proceed automatically when only statement counts are desired, and many others, coupled with the fact that students can use the animation system on their own after class, make this tool indispensable for teaching and learning computer science and programming.

Just what are the prospects for such a program animation system? Excellent! Availability?...well, many already exist, albeit not always in a form entirely suited to beginning students. Some exist as *interactive debuggers* and are part of most modern programming language systems. Other systems, often referred to as *visible programming systems*, are also available; these are better suited to teaching and learning programming but don't include many of the desired educational capabilities described above. (It should be noted here that the term "visual programming" is also used in the literature. It refers to a more general view of programming with visual aids and is not restricted to describing only program animation. The term we use—visible programming system—is used only to describe facets of program animation.) Fully developed

educational program animators with all of the envisioned features presently exist only in test form or as ongoing projects, but are likely to be available soon. (As is true of many good educational ideas, the technology for the implementation of educational program animators is certainly available; what has been lacking are the resources and incentives to bring such a product to market.)

Let's examine some of these systems.

Educational Program Animators

To be successful, an educational program animator must not only incorporate the features described in the previous section, it must also be extremely easy to use. Any system that requires substantial set-up time by teachers (who may be new to computer science to start with) or that demands that users (teachers and students) learn a complex interface will fail.

As already noted, animators that combine the many desired educational features with absolute ease of use are not yet available. However, a pilot system designed by the author with many of these features has been in use at Montana State University for some time now. This system is called DYNAMOD (for DYNamic Algorithm MODerator), and has been described most recently in Ross (1991a). The success of DYNAMOD with both instructors and students of introductory computer science courses has led to a new project supported by the National Science Foundation (Ross, 1991b) for development of a greatly enhanced system called DYNALAB (for DYNAmic LABoratory). DYNALAB will contain all of the features on the wish list of the previous section, and it will support experimentation in formal computer science laboratories. DYNALAB will be useful in introductory computer science courses at both pre-college and college levels.

To provide a tantalizing taste of what an educational program animator can do, a description of DYNALAB is given below. Although DYNALAB is not yet finished, it will be presented as if it were to facilitate the discussion.

DYNALAB. Suppose that accompanying the textbook (or perhaps even replacing the textbook) for an introductory computer science laboratory course there is a standard 3.5 inch disk for either IBM or Macintosh personal computers that contains all of the supporting materials for the course. In particular, the disk contains the following:

1. A comprehensive library of Pascal programs, ready to be run in animated fashion on a computer video screen.
2. A library browser that allows the user to meander through the library, looking up programs by the topics they illustrate (e.g., loops, recursion, searching, sorting, time complexity, etc.).
3. An animation system that allows the user to retrieve any program in the library and run that program in visual, animated form on the computer video screen with minimal and very easy user intervention.
4. A library maintenance routine that allows an instructor to enter more example programs into the library or to modify existing programs.
5. A student section to the library, where the student can enter self-designed programs (or other library programs that have been modified) for possible execution under the animator.

6. A complete set of laboratory experiments that can be performed on programs in the library.
7. A user-friendly interface that provides access to all of these features, including context-sensitive help information that can be accessed from any place in the system.

This is a picture of the DYNALAB system. To see how DYNALAB can be used in a course, a few examples are given below. These scarcely scratch the surface of applications of a program animator in teaching and learning computer science and programming, but they should be enough to whet the appetite.

Understanding Programming Language Constructs. As illustrated in examples 9 and 10, every programming language construct is more clearly explained by way of DYNALAB—from simple assignments statements, to *if* and *case* selection statements, to loops, to procedure and function calls, to parameter passing, and to recursion. One particular experience the author had in using the DYNAMOD pilot animator for the first time graphically illustrates this. In a second-quarter class of programming students who had already completed a few programs using recursion, an animated example of a recursive procedure was being projected on the screen. The students were spellbound, and from the back of the class of about a hundred, a student fairly shouted, "So *that's* how recursion works!"

Determining Program Execution Paths. Most programs include selection statements (*if* and *case* statements, for example). Depending on the data being processed, different paths (selections) will be followed as the program is executed. Having the students come up with different data values that will cause every path through the program to be executed is a good exercise to help them clearly grasp the dynamic workings of the various selection statements. Since the students all have access to the same library with the same animated programs, they can do such an exercise without first having to write and debug the program of the assignment.

Modifying Existing Programs. Since all students have a copy of the animation system, they can be required to modify a particular program in the library as an assignment. This can be as simple as changing an *if* statement to an equivalent *case* statement to as complex as incorporating new procedures and functions.

Empirically Validating Time Complexity. The previous two examples are primarily oriented towards teaching and learning programming. Exercises intended to elucidate some of the fundamental issues of the science of computing are also possible. For example, some programs are difficult to analyze for their time complexity analytically. Such a program can be assigned from the library for which the students are to verify the time complexity empirically. This requires that they determine a number of different appropriate input sets of varying sizes, run the program on each of the input sets, record the statement counts for each, and then analyze the results (perhaps in graph form) to determine what the time complexity is. This kind of assignment has proved to be extremely effective in teaching and learning about time complexity. Again, since all students have the same, correct programs to work with, their experiments are repeatable and easy to verify, as any good science experiment intended for the classroom should be.

Library Customization. As new experiments or examples are envisioned, these can be easily incorporated into the library and new diskettes distributed to the students, making the DYNALAB system adaptable to different situations and teaching/learning styles.

Program Review. Finally, it should be noted that just observing one's own program in action is a rewarding and educational experience. Students who have access to an animator for running their programs can't resist watching them run under the animator. This seems to be a universal observation: what artisan doesn't enjoy examining his or her creation in depth and from new and different angles?

In summary, the advent of easy-to-use educational program animators promises to make the teaching and learning of programming and introductory computer science fundamentals much more convenient and easy. Of particular interest to teachers is the fact that an educational program animator like DYNALAB is a complete resource: all programs, textual material, exercises, and assignments necessary for supporting a course in introductory programming and computer science are incorporated into the system on a diskette. If teachers desire to install their own programs and exercises, they may, but otherwise they are free to concentrate on teaching rather than program and assignment development. Students, in turn, will find the use of the animator to be straightforward, requiring almost no training in the use of the system. These are among the features that distinguish educational program animators from visible programming systems and interactive debuggers, described next.

Interactive Debuggers

Interactive debuggers are the most common type of program animation systems. All good programming language systems come with an interactive debugger, because as a tool for ferreting out errors in a program, a good debugger is indispensable. Even correct programs, when examined through the animation of an interactive debugger, can be better understood and perhaps improved.

Interactive debuggers provide program animations similar to those described in the previous section. The primary difference between interactive debuggers and educational program animators is that the animations are not automatic in a debugger. The user is responsible for most aspects of the animation: the variables to be displayed during the animation must be chosen specifically, a complex (for beginners) interface for driving the animation must be learned, and various options for configuring the animation must be mastered. The reason for this complexity is simple: interactive debuggers are for professional programmers, those who understand programs and programming well, and who are trying to uncover bugs or analyze program behavior for improvements; the tool must be up to the task. Among the common features of an interactive debugger are these.

- ***Breakpoints.*** A code, called a breakpoint, can be inserted into a program at an arbitrary position. The program will execute at full speed until reaching this point and then pause. At this point the user can step through the program a line at a time as the animation unfolds on the screen. The purpose of a breakpoint is, of course, to allow examination of only sections of code that contain errors or are otherwise of interest.

- ***Memory Observation.*** The user can specify a set of variables and expressions that are of interest for observation. The selected variables and their values will be displayed by the animator during the course of a session.

- ***Reverse Execution.*** The user can request that execution reverse at any point, thus allowing review of a section of code numerous times without restarting rhe program (only a few of the best debuggers have this feature).

- ***Profiling.*** Statistics on how often a specified line, section of code, or procedure is executed can be generated, along with numerous other statistics that characterize the execution of a program. These can aid in program time and space complexity analysis.

In summary, a full-featured interactive debugger brings powerful tools to bear on program debugging and analysis. These tools are appropriate for advanced students, but not beginners just learning programming. Teachers can certainly use an interactive debugger to illustrate program dynamics in class quite effectively. However, designing, writing, testing, and presenting the programs are all the responsibility of the teacher, unlike the case for an educational program animator, where complete programs are already provided. Also, since interactive debuggers are designed for debugging rather than teaching, a number of nice features for teaching are lacking. A simple example is the pause feature described in the provious section. Instead of markers (?==> and ==>) that pause at the same statement both before and after execution of a statement to allow consideration of both what will happen when the statement is executed and then where control will go next, a statement is executed and control passes to the next statement in one indistinguishable move. So, while interactive debuggers can be pressed into service for program animation, they are not as useful in introductory courses as an educational program animator.

Visible Programming Systems

Visible programming systems bear mentioning that they are a slight variation on interactive debuggers. Aimed more at beginning programmers, visible programming systems allow a program to be run in "visible mode." In our terminology this just means that program animation can be turned on when a program is executed, and a display similar to those given in the section "Educational Program Animators" in this chapter appears on the screen. Visible mode is generally automatic; the user does not need to orchestrate the animation to such a degree as is required of professional debuggers. Again the reason for this is that a visible programming system is aimed more at those learning programming than advanced programmers, whereas the users of interactive debuggers are generally professional—or at least advanced—programmers. A good example of a visible programming system is Dr. Pascal (1989).

Beyond program animation, visible programming systems do not have many of the pedagogical features of educational program animators. For example, they do not come with libraries of typical programs animated for class use, they generally ignore time complexity measures, and they do not imcorporate exercises and experiments. However,

they certainly represent a vast improvement over non-animated systems for teaching and learning programming and should be used in the absence of full-featured educational program animators.

History and Sources of Program Animation Systems

The entire topic of visualization with respect to programming is a large one. Good references for an overview are Shu (1989), Chang (1990), Glinert (1990), and Ichikawa et al. (1990) (the latter two include papers on algorithm animation). Program animation specifically has intrigued programmers from the outset, although it was not until video screens became available that animations reached their full potential. The early systems were, not too surprisingly, envisioned as debugging aids for working programmers. Early references to such systems include Evans and Darley (1966), Balzer (1969), Barron (1971), and Cuff (1972).

Educational program animators began to receive attention in the literature in the early 1980s, including Rezvani and Ross (1981), Ross (1982), Ross (1983), and Hille and Higginbotham (1983). Algorithm animation, the subject of the next section, was also receiving more interest during this period, and a slight confusion of terminology exists in the literature; what we refer to in this chapter as algorithm animation was often called program animation. It is usually immediately clear from the context of the paper which type of animation is intended, however.

As educational program animators (e.g., Ross 1991a, 1991b; Mayerhofer & de Lucena, 1992) have not yet appeared as commercial ventures, they are difficult to obtain, or—being pilot projects—are not yet as convenient to use as they could be. Most, like the DYNALAB system described previously, are ongoing research and development projects at universities and do not enjoy the wide exposure and availability of a commercial project. Nonetheless they hold great promise for enhancing introductory computer science courses and should soon become widely available. In the meantime, visible programming systems, such as Dr. Pascal (1989), can be used to great effect in introductory programming courses, both by teachers, who can display animations on wall screens during class discussions, and by students, who can watch their own programs in animated form. (Dr. Pascal runs on IBM PCs.) With a bit more effort, the debuggers that come with most programming language systems can be pressed into use. Examples include the debugger that comes with Turbo Pascal for IBM PCs and the Think Pascal system for Macintoshes. Others with similar features are available.

Algorithm Animation

Algorithm animators provide a different—but complementary—visualization technique for computer science education. Instead of animating the *program* that implements an algorithm, the *algorithm* itself is animated. In other words, it's the *concept* that is animated, not the statements of some program.This is explained in the next section.

Making Algorithms Come to Life

The ideas behind algorithm animation can best be explained by example.

Example 11

Consider the problem of sorting. Pictorially, the values to be sorted can be represented on a computer screen by bars of various heights (the taller the bar, the larger the value represented). Initially, in representing a list of values in random order, the bars themselves could be pictured on the screen in random order, as shown in Figure 1.

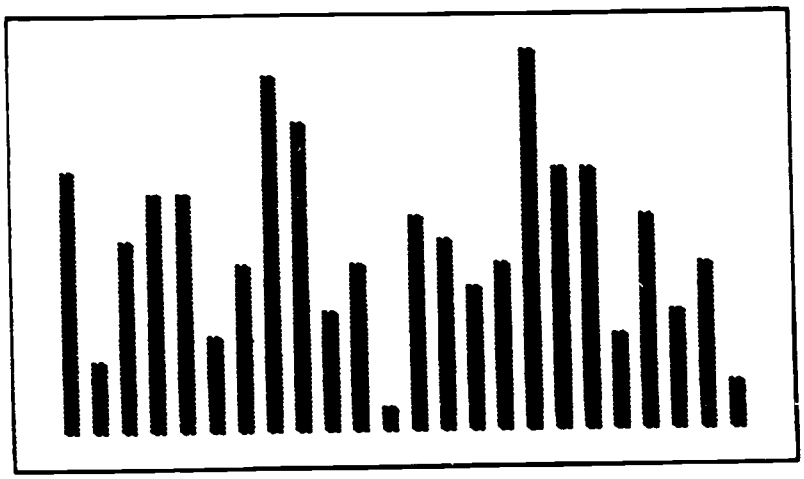


Figure 1. Visualizing a random assortment of numbers

The purpose of sorting is to rearrange a set of values so that they appear in, say, ascending order. In the above visualization, this would mean that the bars should be rearranged to be displayed from shortest to tallest from left to right across the screen. Notice that the bars are only representative of the actual values that a sorting algorithm might be applied to. The actual values could be numbers, names, or even entire records of information. There are many different sorting algorithms, and although each would achieve the same objective of rearranging the values into ascending order, each would do this through a different strategy. The animation of an algorithm should clearly demonstrate the strategy of the algorithm.

Here is one possible algorithm for sorting, called *insertion sort*: Start with the first value (bar) in the list and proceed through the list (from left to right in the picture). For each new bar encountered, compare it to the previous bar. If the new bar is shorter, switch the two bars. Continue this process of comparing this new bar with the previous bar and switching the two until the new bar has arrived at its proper position in the list (the bar to its left is shorter).

Suppose that the first five bars in the list have been handled this way already and we are about to look at the sixth bar. At this point the visualization should look as shown in Figure 2. The first five bars of the list are already in ascending order as we are about to begin processing the sixth bar. The sixth bar is marked differently to denote that it is the one currently being examined.

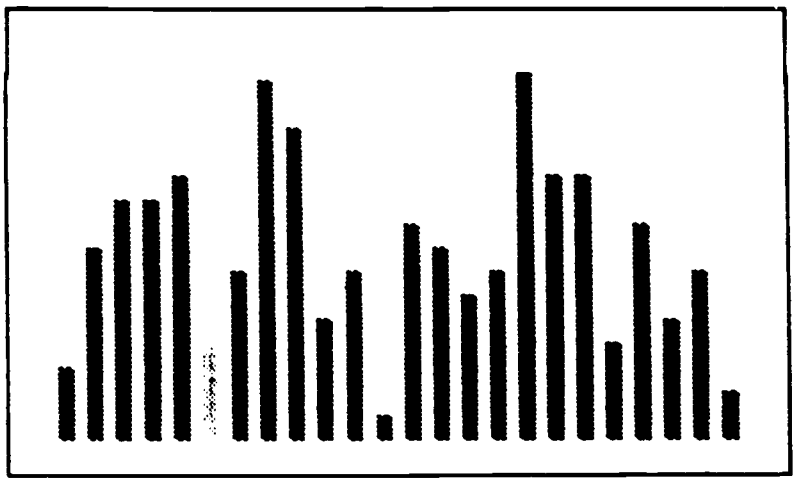


Figure 2. Examination of the sixth bar

Now compare the new bar (the one marked differently) with the one to its left. Since the marked bar is shorter, the two bars are switched, yielding Figure 3.

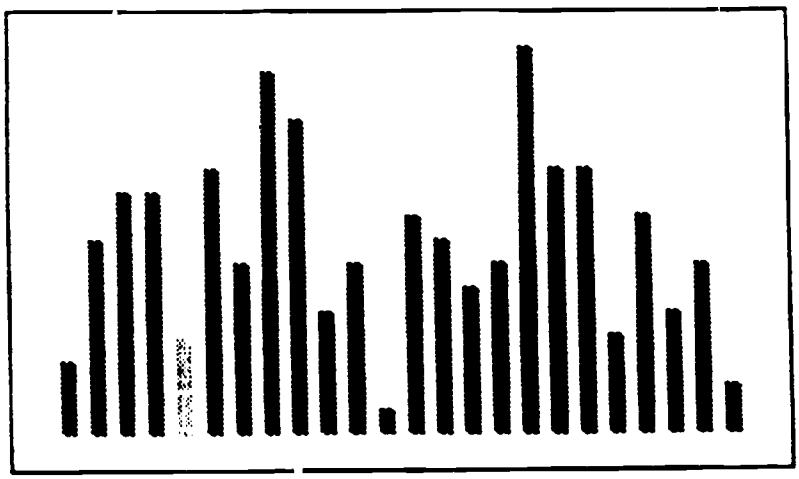


Figure 3. Result after switching the fifth and sixth bars

Again, the marked bar is compared with the one to its left, and since the marked bar is shorter, these two are switched as well, resulting in Figure 4.

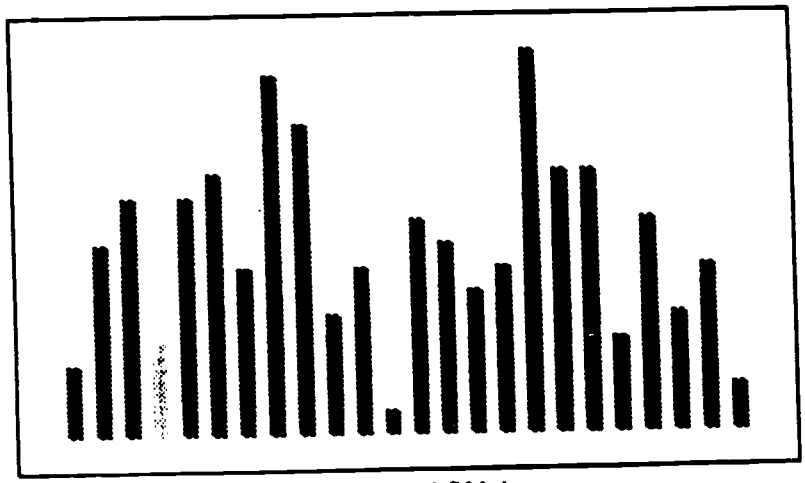


Figure 4. Result after switcing the fourth and fifth bars

This process continues until the marked bar arrives at the second position, where it is no longer shorter than the bar to its immediate left, as shown in Figure 5.

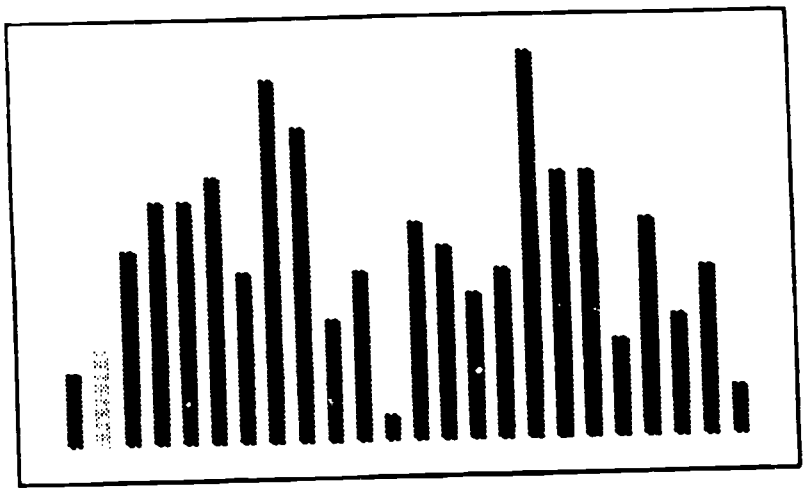


Figure 5. Final result of moving the marked bar left

At this point, the first six bars are in ascending order, so the seventh bar is marked as shown in Figure 6 as the new one to be examined, and the process repeats.

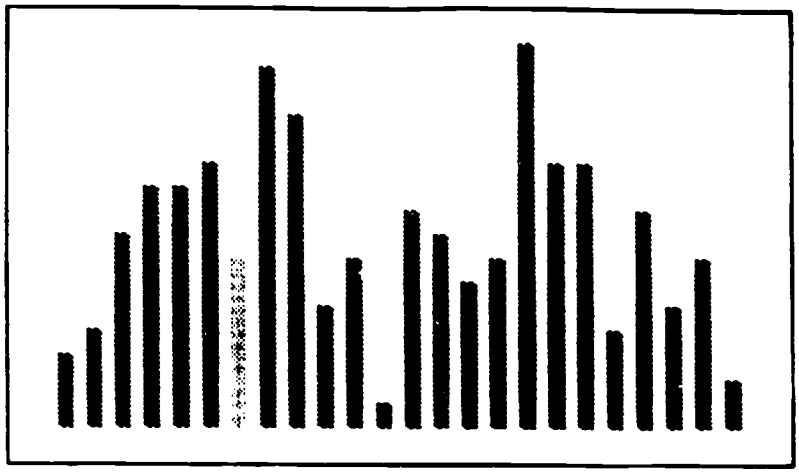


Figure 6. Examination of the seventh bar

As can be seen, this algorithm will eventually lead to the entire set of bars being sorted into ascending order by height. An algorithm animator would allow a user to watch the sorting process one step at a time, controlled by keystrokes, or perhaps continuously without intervention. In any case, this method of sorting would become graphically apparent to the viewer. The use of colors for the bars as well as options to speed up or slow down the animation would greatly enhance the presentation. ■

As this example illustrates, a visual animation of an algorithm can lead to a clear understanding of the algorithm. As a tool for teaching and learning about algorithms, algorithm animators are indispensable. (If you aren't already convinced, try describing the actions of an algorithm on a blackboard or in textbook form as in example 11 above; you'll soon become a believer!)

Comparing Algorithms

In section 2 it was pointed out that one of the key issues of computer science is the classification of problems and algorithms according to their time complexities (how efficient they are). Algorithm animators provide teachers and students an excellent opportunity to compare algorithms for efficiency, thus reinforcing this important aspect of the science. They also allow for scientific experimentation in algorithm analysis.

The problem of sorting is a particularly good example, because there are so many different sorting algorithms. In the classic textbook, *Sorting and Searching* (Knuth 1973), twenty-five different sorting algorithms are discussed, which only represent a fraction of the methods devised by that time. While many sorting methods that have been developed are now obsolete (other, more efficient methods have been discovered), many others fulfill a useful role. How can students visualize the differences among so many choices, and how can they be convinced that some are better than others? Algorithm animators provide an effective answer.

Example 12

Consider the problem of comparing four different sorting algorithms. An algorithm animator could open up four different windows on the screen, each with the same set of bars in the same random order to start with. Each algorithm will eventually sort the bars into ascending order, and so each window will also end up with the same picture as well. In between, however, each algorithm will rearrange the bars individually in completely different manners, so the pictures will be quite different.

The animator can be set to cycle through the windows, doing just one operation in each of the four windows in succession. The result will be that the viewer can see how each algorithm is progressing compared to the others. Good algorithms will finish the sorting process sooner than the less efficient ones, clearly demonstrating their benefits.

The user can study the effects of algorithms on different data sets to see what the results are. For example, if the list of values to be sorted happens to already be in order, how do different sorting strategies compare? Some that are very good on randomly ordered values, such as the *quicksort* algorithm, do very poorly if the initial list is already in order or is in reverse order. On the other hand, the insertion sort algorithm given in example 11 is not nearly as efficient as the quicksort algorithm on randomly ordered lists, but is much faster on a list that is already in order. This ability to experiment with various algorithms within the framework of an animation system greatly enhances the learning process. ■

History and Sources of Algorithm Animation

References to algorithm animation indicate that this idea was being explored at least by the late 1960s. In a nice paper that summarizes these activities (Baecker, 1975), Baecker describes the approaches taken. As there were no personal computers and graphics terminals available, movie films were developed that animated algorithms. The work required to develop such a film was apparently quite incredible, and of course, there was no way for the viewer to interact or experiment with the animation. One result of these efforts was the film *Sorting Out Sorting* (Baecker & Sherman, 1981) that animated various sorting algorithms for comparison. These early researchers recognized that the technology for developing interactive, computer-based animations was just around the corner, and not many films were produced.

Since the mid 1980s a number of computer-driven algorithm animators have been developed. These include the work of Marc Brown (Brown 1988a, 1988b, 1991; Brown & Hershberger, 1991; Brown, Meyrowitz, & van Dam, 1983; Brown & Sedgewick, 1984, 1985), Robert Duisberg (1986, 1988), Esa Helttula et al. (1989), John Stasko (1990), and Thomas Naps (1990). International interest in both program and algorithm animation has flourished, as well; the Algorithm Simulation and Animation Environment (ASA) under development in Brazil is indicative (Mayerhofer and de Lucena 1992). Others have designed similar, more limited algorithm animators that have not been published widely. Probably of most interest to teachers of introductory computer science courses are the following algorithm animators: Balsa II (Brown, 1988a) which runs on Macintosh computers; Tango and XTango (Stasko, 1990), which run on workstations (the latter requiring the X-window system); and GAIGS (Naps, 1990), which runs on IBM PCs. Other systems also exist, as mentioned, but these seem to be the most advanced and current.

Algorithm animators specifically designed for education are in a more advanced state than are similar program animators, but they both suffer from a common problem: availability. As far as the author knows, none of these systems is available commercially or as part of an instructional package. Most are the result of research done in teaching and learning computer science at universities and thus exist primarily as test projects. Therefore, software and technical support—not to mention courseware—are likely to be lacking. Nonetheless, algorithm animators can be used to great effect in the classroom, and it should be only a matter of time before they find their way into integrated course materials.

A final note on algorithm animators is this: installing new animations of different algorithms into the system can be quite complex. From the user's perspective, viewing animations that are already included is easy; trying to devise a new animation (e.g., from the perspective of a teacher who would like to demonstrate something new) is the difficult part. What kinds of illustrations would graphically represent the algorithm and its actions best? How can these illustrations be animated, perhaps with sound and color to achieve the greatest success? These are the challenging questions to answer in developing successful algorithm visualizations. However, this should not deter teachers from using algorithm animators, as most come with a sufficient number of relevant animated algorithms for most introductory computer science courses.

Other Prospects for Computer Science Visualization

Computer generated animations and visualizations can be very effective in specific topic areas of computer science as well. How jobs are processed in a multiprogramming operating system, how the stack progresses during compiling, how a network handles packets, how a Turing machine or finite state automaton works, how the instruction fetch and execute cycle of a computer operates in the presence of interrupts, and so forth are all good prospects for visualization (indeed, some have been tried). In this chapter we have described just those visualization systems that are at the core of computer science, namely algorithmics.

Work continues on visualization techniques for this core area of computer science. In an interesting paper entitled "Color and Sound in Algorithm Animation" (Brown & Hershberger, 1991) work is described that incorporates color and sound into an algorithm animation system. Work also progresses on making algorithm animators more general and easier for the user to incorporate new animations. We can expect that as program and algorithm animation systems mature, we will find more of them incorporated into courseware available to introductory secondary school and university computer science courses.

Summary

As described in this chapter, scientific visualization techniques that support education in the fundamentals of computer science—namely algorithmics—fall into two main categories: program animation and algorithm animation. Among the program animation variety are educational program animators, visible programming systems, and interactive debuggers. Interactive debuggers can be used for animating programs

for teaching and learning introductory computer science and programming, but they are tools designed for the programmer rather than the educator and beginning student. Substantial effort is required on the part of the teacher to develop programs, exercises, experiments, and textual material in support of a course if one of these types of systems is employed. In addition, the student must master a rather complex interface to truly benefit from using one of these systems. Visible programming systems are similar to interactive debuggers in their orientation towards program debugging. They are, however, generally much easier to use and more readily adaptable to assignments and discussions involving beginning students.

Educational program animators, on the other hand, provide a complete resource: programs, exercises, experiments, and textual material are all included within the framework of an easy-to-use program animation system. The teacher need only design the course around the animator without being required to expend the considerable effort necessary to develop programs and accompanying assignments (as would be the case if an interactive debugger or visible programming system were used). Educational program animators do, however, allow the teacher to include a few favorite exercises as desired. Students benefit from having access to the same resources as the instructor, providing consistent assignments and experiments that are repeatable. In addition, the user interface is designed to be useable by the utter novice, making the animations accessible to all students.

Algorithm animators go a step beyond program animators by visualizing the ideas behind the algorithms in some kind of analogous, graphical form. Animating an algorithm requires a lot of effort and careful design, whereas program animation occurs automatically. Program animation and algorithm animation complement each other well and will eventually be combined into comprehensive computer science visual education packages.

For the computer science teacher of today, educational program and algorithm animators offer a quantum leap in teaching and learning capabilities. In fact, one could even say that science classrooms in general are going to be remodelled to include windows to many breathtaking views of science never seen before. The windows will be personal computers and workstations, and the views will be computer-generated visualizations of scientific principles and objects. One can only hope that these views will bring students to see the exciting possibilities in science and mathematics and inspire more of them to aspire to careers in these fields.

References

Baecker, R. (1975). Two systems which produce animated representations of the execution of computer programs. *ACM SIGCSE Bulletin, 7*(1), 158-167.

Baecker, R., & Sherman, D. (1981). *Sorting out sorting*. University of Toronto Computer Media Centre, 121 St. George St., Toronto, Ontario, M5S 1A1.

Balzer, R. (1969). EXDAMS—Extendable Ddbugging and monitoring system. *AFIPS Conference Proceedings, 34*, 567-580.

Barron, D. (1971). Approaches to conversational fortran. *The Computer Journal, 11*(2), 123-127.

Brown, M., Meyrowitz, N., & van Dam, A. (1983). Personal computer networks and graphical animation: Rationale and practice for education. *ACM SIGCSE Bulletin*, *15*(1),296-307.

Brown, M., & Sedgewick, R. (1984). A system for algorithm animation. *Computer Graphics*, *18*(3),177-186.

Brown, M., & Sedgewick, R. (1985). Techniques for algorithm animation. *IEEE Software*, *2*(1),28-39.

Brown, M. (1988a). *Algorithm animation*. Cambridge, MA: The MIT Press.

Brown, M. (1988b). Exploring algorithms using Balsa-II. *Computer*, *21*(5),14-36.

Brown, M. (1991). Zeus: A system for algorithm animation. *Proceedings of the 1991 Workshop on Visual Languages*.

Brown, M., & Hershberger, J. (1991). *Color and sound in algorithm animation* (Research Report 76a). DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301.

Chang, S. (Ed.). (1990). *Visual languages and visual programming*. New York: Plenum Press.

Cuff, R. (1972). A conversational compiler for full PL/I. *The Computer Journal*, *15*(2), 99-104.

Denning, P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, A., & Young, P. (1989). Computing as a discipline. *Communications of the ACM*, *32*(1), 9-23.

Dr. Pascal. (1989). *Dr. Pascal user manual*. A Pascal programming language system from Visible Software, P.O. Box 7788, Princeton, NJ.

Duisberg, R. (1986). Animated graphical interfaces using temporal constraints. *Human Factors in Computing Systems: CIII '86 Conference Proceedings*, 131-136.

Duisberg, R. (1988). Animation using temporal constraints: An overview of the animus system. *Human-Computer Interaction*, *3*, 275-307.

Evans, T., & Darley, D. (1966). On-line debugging techniques: A survey. *AFIPS Conference Proceedings*, *29*, 37-50.

Glinert, E., (Ed.). (1990). *Visual programming environments: Applications and issues*. Los Alamitos, CA: IEEE Computer Society Press.

Harel, D. (1992). *Algorithmics: The spirit of computing* (2nd ed.). Menlo Park, CA: Addison-Wesley.

Hartmanis, J., & Lin, H. (Eds.). (1992). *Computing the future: A broader agenda for computer science and engineering*. National Academy Press.

Helttula, E., Hyrskykari, A., & Raiha, K. (1989). Graphical specification of algorithm animation with ALLADIN. *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, *11*, 892-901.

Hille, R., & Higginbottom, T. (1983). A system for visible execution of pascal programs. *The Australian Computer Journal*, *15*(2), 76-77.

Ichikawa, T., Jungert, E., & Korfhage, R. (1990). *Visual languages and applications*. New York: Plenum Press.

Knuth, D. (1973). *Sorting and searching*. Menlo Park, CA: Addison-Wesley.

Merritt, S. (Moderator). (1992). ACM model high school computer science curriculum (panel discussion). Twenty-third SIGCSE technical symposium on computer science education. *SIGCSE Bulletin*, *24*(1), 323.

Mayerhofer, M., & de Lucena, C. (1992). Design of an agorithm simulation and animation environment (ASA). *SIGCSE Bulletin*, *24*(2),7-14.

Naps, T. (1990). *GAIGS for the PC: User's manual*. Thomas L. Naps, Lawrence University, Appleton, WI 54912.

Rezvani, S., & Ross, R. *A dynamic library of interactive programming language examples* (Tech. Rep. CS-81-073). Computer Science Department, Washington State University, Pullman, Washington 99164.

Ross, R. (1982). Teaching programming to the deaf. *ACM SIGCAPH Newsletter, 30*,18-24.

Ross, R. (1983, Fall). LOPLE: A dynamic library of programming language examples. *ACM SIGCUE Bulletin*, 27-31.

Ross, R. (1991a). Experience with the DYNAMOD program animator. *Proceedings of the Twenty-Second Technical Symposium on Computer Science Education, SIGCSE Bulletin, 23*(1), 35-42.

Ross, R. (1991b). *A dynamic computer science laboratory*. Project funded by the National Science Foundation, NSF Grant Number USE-9150298.

Shu, N. (1989). Visual programming: Perspectives and approaches. *IBM Systems Journal, 28*(4), 525-546

Stasko, J. (1990). Tango: A framework and system for algorithm animation. *IEEE Computer, 23*(9), 27-39.

Taylor, H., & Martin, C. (1992). The impact of new accreditation and certification standards for secondary computer science teachers on university computer science departments. Twenty-Third SIGCSE Technical Symposium on Computer Science Education *SIGCSE Bulletin, 24*(1), 235-239.

Tucker, A. (Ed.). (1991). *Computing curricula 1991*. Report of the ACM/IEEE-CS Joint Curriculum Task Force. ACM Press, ACM order number 201910.

Weiss, M. (1992). *Data structures and algorithm analysis*. The Benjamin Cummings Publishing Company.

Acknowledgement: The DYNALAB project described in this paper is based in part upon work supported by the National Science Foundation under Grant No. USE-9150298. The Government has certain rights in this material. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Chapter 7

Getting Started With Supercomputing: An Approach for High School Students

DONALD W. HYATT

Because of remarkable advances in computer technology, modern scientists have a new problem solving tool, a supercomputer. Supercomputers supported by high powered graphics work tations have become extremely valuable resources for a wide range of current scientific investigations. High performance computing will become increasingly more important to many scientific and engineering disciplines, so it is important for educators to prepare future generations to be ready for that growing demand.

Supercomputer access was originally reserved for graduate students and research scientists. Now, however, this power is available to high school students through a national competition called SuperQuest. This contest is an exciting educational initiative promoted by many of the national supercomputing centers including the Cornell Theory Center, the National Center for Supercomputing Applications, and the University of Alabama/Alabama Supercomputing Network. It is jointly sponsored by the National Science Foundation and various private corporations. To be ready for the technical expertise that will be required in the next century, both students and teachers must learn to use this emerging technology. As a teacher who has been involved with SuperQuest since the first competition in 1988, the author hopes that the following materials will be helpful to other educators who want to get started with supercomputing.

Background

Throughout history, scientists have been searching for the fundamental laws which govern our universe. Dr. Clifford N. Arnold (1988), former manager of Computational

Research at ETA Systems, pointed out to participants of the first SuperQuest that there are now essentially four fundamental methods of scientific research.

Until recently, only three of these techniques had been used for most scientific investigations. The first method, observational science, is where researchers study some situation or phenomenon, and then carefully document their discoveries. Some examples of observational science include studying the social behavior of gorillas in various habitats, or mapping the geological formations in the Grand Canyon.

The second method is referred to as experimental science. With this technique, an experiment is designed that will provide some insight into a basic scientific principle. It is important in experimental science to have control groups for comparison, and to try to hold many factors constant in order to isolate cause and effect. Examples of experimental science include tests to determine the appropriate concentrations for a new medication, or comparative tests of airplane wing designs in a wind tunnel.

In the third method, theoretical science, a law or theory is hypothesized and then substantiated by additional research and rigorous mathematics. Examples of theoretical science include the complex equations describing fluid flow, and the familiar formula, $e=mc^2$, in Einstein's theory of relativity.

Computational Science

The fourth and newest method of scientific investigation uses advanced computer technology and is referred to as computational science. Because high performance computers have become so powerful, scientists are now able to use them as tools to study a wide variety of very complex problems. Supercomputers can help devise safer automobiles, build better airplanes, create special effects in Hollywood movies, and can even be used to design faster supercomputers.

A very practical application of supercomputer power is in the area of computer modeling and simulation. One distinct advantage of certain computer models is that they can be used to speed up extremely slow processes in order to predict potential outcomes in the future. Long range weather forecasting and the problems of global warming are environmental studies which are being investigated by supercomputer models. Scientists have even been able to study the effects of seemingly harmless forest management techniques which have now contributed to increased forest fire potential.

Computer simulation models are also excellent choices to study processes which happen too quickly to observe by a direct experiment. They are very useful to investigate the behavior of things which might be too small to examine by any physical means. Some examples include molecular dynamics simulations and three dimensional modeling of chemical compounds which can help scientists understand the physical properties of these structures.

Computer models can be used to explore situations which researchers cannot experience directly. For instance, some scientists are investigating the nature of black holes using supercomputer simulations while others are studying the development of severe thunderstorms in order to understand the internal conditions which might cause tornados.

Graphics Visualization

Closely related to supercomputing is the field of graphics visualization. Instead of just looking at numbers, scientists can display supercomputer output in a visual form

using high quality computer graphics. Through computer graphics, researchers are often better able to comprehend large quantities of data and notice subtle trends. In the excellent series NCSA RealTime (1992), the National Center for Supercomputing Applications has compiled some motivational video segments which show the wide range of graphics visualization techniques being used in current supercomputer applications.

Some graphics visualization approaches may use just simple two-dimensional graphs of lines or dots on the screen, while others can display three-dimensional renderings of objects, contours, and surfaces. Advanced visualization techniques may include computer animations with realistic colors, reflections, and shading (Foley, van Dam, Feiner, and Hughes, 1990).

Color is a very valuable tool in graphics visualization. It can be used to enhance images or provide emphasis to details which might not be readily apparent. Engineers can better view air currents and turbulence around new car designs with color enhancement. Architects can look for regions of material stress in structures which might fatigue under heavy use.

Doctors are using three-dimensional imaging to view cancerous tumors in the body which might be very difficult to discern by conventional x-rays. Using similar techniques, anthropologists can even investigate the interior of ancient mummies without ever unwrapping them.

Supercomputing Applications for High School Students

Students at the secondary school level should realize that many of the previously mentioned applications using supercomputers involve huge projects with teams of scientists, technicians, and programmers. The mathematics being used in some of these investigations is often far beyond the scope of a typical high school curriculum. However, there are computational science techniques which can be incorporated into the typical high school program. Some methods which have been used successfully at Thomas Jefferson High School for Science and Technology are presented in the following examples.

Example #1 - Missile Trajectory

This example investigates a standard physics problem of missile trajectory, the typical scenario of a projectile launched toward a target which is a known distance away. The problem is to find which combinations of launch angle and corresponding velocity would be needed in order for the projectile to land arbitrarily close to that target. However, instead of solving the problem analytically, the following presentation uses a computational science approach.

Equations of the Model

As shown in Figure #1, the height y of the projectile at time t is defined as:

$$y = v_y * t - 0.5 * a * t^2$$

where v_y is the component of velocity in the vertical direction and a is the acceleration toward earth by the force of gravity.

The horizontal distance x travelled during time t is defined by the simple equation:

$$x = v_x * t$$

where v_x is the component of velocity in the horizontal direction.

Given the initial velocity v and launch angle theta, the components of velocity can be obtained using simple trigonometry:

$$v_x = v * \cos(\text{theta})$$
$$v_y = v * \sin(\text{theta})$$

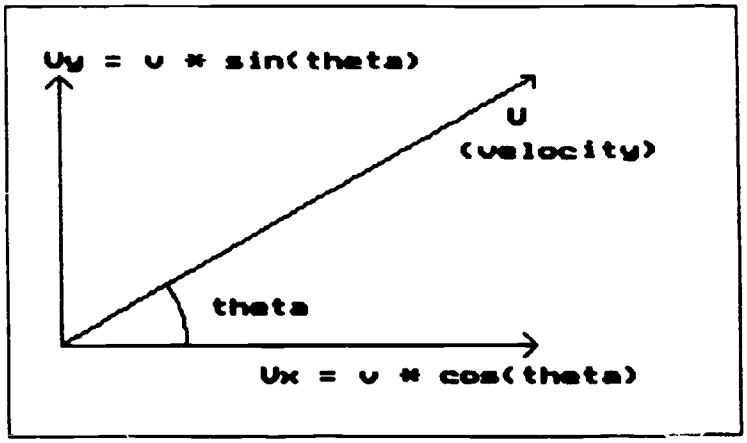


Figure 1. Diagram of basic model parameters

Model Prototype

The first step in creating a computational model is to develop a simple prototype and then validate it against known behavior. For this example, a possible first stage might be to display a simple graphic showing the position of a projectile over time given an initial velocity and launch angle. The number of points plotted on the screen depends upon the time step, or the length of time between successive values of t. The following pseudocode describes the basic algorithm.

Enter Values:
- v (velocity),
- theta (launch angle),
- dist (target distance),
- range (error margin or acceptable radius for "hit")

Initialize Variables:
- t=0 (time),
- tstep = 0.1 (time step between calculations)
- a=32 (Gravitational constant in ft per sec^2)

Find velocity components:

$v_x = v^* \cos(\text{theta})$

$v_y = v^* \sin(\text{theta})$

Loop through all combinations:

Repeat

$t = t + \text{tstep}$ (Increment time step)

$y = v_y * t - 0.5 * a * t^2$ (Calculate height)

$x = v_x * t$ (Calculate horizontal distance)

plot(x,y) (Plot position of projectile)

Until $y <= 0$ (Until missile has hit the ground)

Test for accuracy:

If Abs(x-dist) < range

then result = Hit

else result = Miss

Figure 2 shows the plot of a projectile shot at two different velocities but at the same angle of 60 degrees. The distance to the target for the simulation is 600 feet, the error margin was 20 feet, and the time step was 0.2 seconds. Notice that a projectile fired at 150 ft/sec would hit the target whereas an attempt at 140 ft/sec would fall short of the mark.

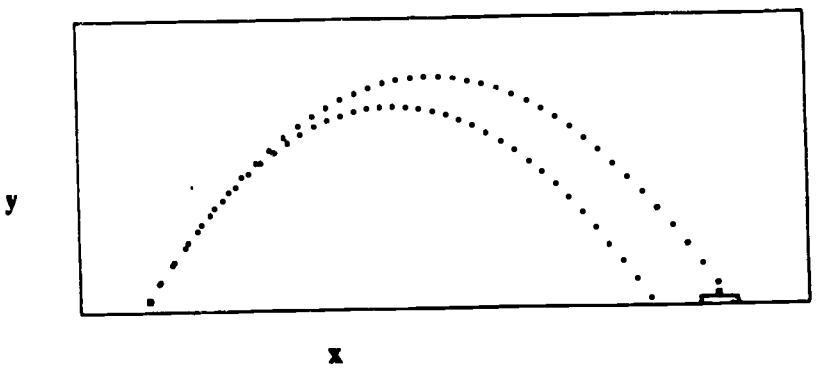


Figure 2. Trajectories at 60 degrees: 140 ft/sec and 150 ft/sec

Validation of the Prototype

Validation of a computer simulation model is one of the most important steps of any investigation. No matter how intricate a model might be, if that model does not adequately define the situation, there is no sense in drawing any conclusions from the simulation. In all computer models, there are certain simplifying assumptions that must be made since it is not possible to accurately simulate every single detail. Therefore, it is important that these simplifications do not interfere with the integrity of the model.

In this example, one possible behavior to validate is that the flight path of the projectile follows an expected parabolic curve. This can be done graphically. Another detail to establish is an appropriate time step between consecutive values of t in order to have predictable results when the projectile finally hits the ground. The time step could be made extremely small to increase accuracy, but that would make the execution

time of the simulation much greater. Supercomputers are able to calculate at very rapid speeds, but there is no sense in wasting computer cycles unnecessarily.

To be sure that the model works correctly, it should be tested with several combinations of parameters where the behavior can be carefully analyzed and compared with known results. Good values to check are examples from mid-range as well as the extremes. Deviations from known behavior must be studied carefully and any problems corrected if the results are to be meaningful.

Expanding to a Supercomputer Application

Modern supercomputers are able to compute at hundreds of millions to even billions of calculations per second. Yet, unless algorithms are designed properly, much of that potential will never be accessed. The key to unleashing the computational power of most supercomputers is to find a way to do many similar calculations in parallel. Some supercomputers use a technique called vectorization which handles arithmetic with large arrays using a method similar to an assembly line. In other supercomputers, the same primary calculation is done by many separate processors simultaneously, but with different initial values.

The basic prototype simulation runs easily on a microcomputer and certainly does not require a supercomputer. However, to make this a supercomputer application, the problem can be expanded to compare a full range of velocities between 0 and 1000 feet per second with all possible launch angles between 0 and 90 degrees. That will require a lot of calculations, but a supercomputer should be able do much of the work in parallel in order to produce results more rapidly. The desired outcome showing a complete picture of acceptable combinations is very computationally intensive, but just right for a supercomputer.

Displaying Results

To analyze the results, it would be possible to print a list of ordered pairs that are solutions, but a better approach might be to select a visualization technique using graphics. The method chosen for this example uses each pixel position on the screen to represent a different combination of distance and launch angle. Horizontal pixel coordinates stand for different velocities and vertical coordinates represent variations in launch angle. There are many combinations to try, and thus, very many simulations to run. In standard VGA graphics, there are 640 pixels horizontally by 480 pixels vertically, or 307,200 combinations. A supercomputer may be able to do many of these calculations simultaneously, although a standard PC would have to try each combination separately.

For each of the defined positions, the prototype model described earlier must go through its full simulation and then decide whether the combination of parameters produced a hit or miss. The screen position is then colored differently depending upon the result: a darker shade for a hit, or two alternating lighter shades for a miss.

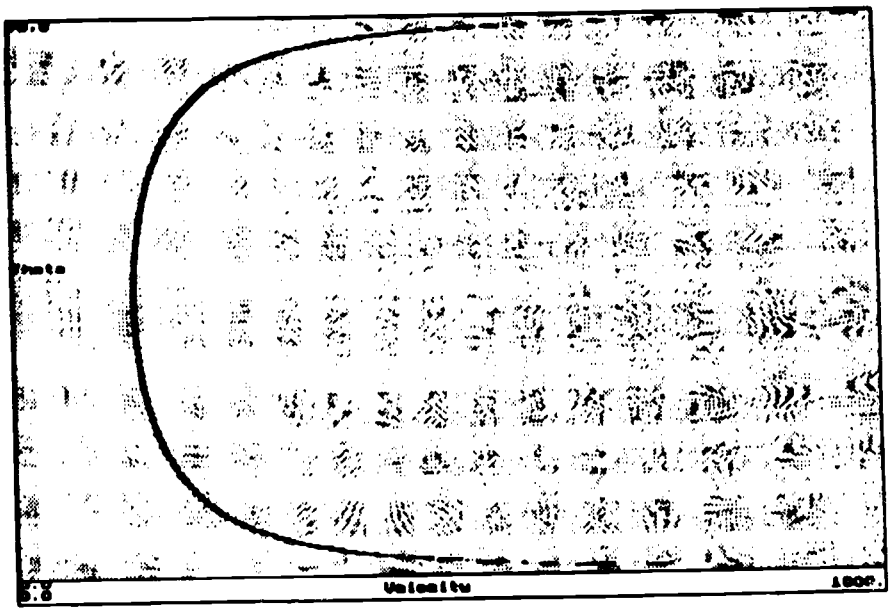


Figure3. Trajectory model computational science solution

The graphic image in Figure #3 shows the results for the expanded model. The horizontal axis represents velocity, the vertical axis represents launch angle, and the solution set is clearly visible by the dark curved line. However, the simulation took several hours to complete on a PC running Turbo Pascal, and any enhancements would make execution time even longer.Therefore, it is a perfect candidate for a supercomputer application. An interesting byproduct of the graphics visualization process in this example is the unusual pattern in the background. The design happens because the simulation plotted alternating colors for various distances where the projectile missed the target. The pattern is a function of not only how often the color changes (every foot versus every ten feet), but also the value used for the time step in the simulation.

Comparison to the Analytical Solution

There is also a purely mathematical solution to this investigation which is not too difficult to derive. Using the basic equations of the model combined with the fact that when the projectile strikes the target, the vertical height is zero yet the horizontal length equals the distance from source to target, it is possible to solve a system of equations which can be resolved into a single form defining velocity in terms of launch angle, given various constants for target distance and gravitation. That relationship can be graphed to show the same solution set described by the computational approach. The mathematical solution is defined by the following equation:

$$v = SQRT((a * dist)/(2 * SIN(theta) * COS(theta)))$$

So why use a computational model when mathematics is obviously more exact? For one reason, if there is desire to expand the model to include other factors such as wind

BEST COPY AVAILABLE

resistance, spins, bounces, and interference with various obstacles, then an analytical approach becomes increasingly more difficult. However, modifications to the computational model are comparatively easy since only additions to the basic simulation need to be defined.

There are times when the computational model may be the only alternative. In fact, some situations which seem relatively simple cannot be solved analytically at all. During the first SuperQuest contest, one student (Scheirer, 1988) from the Jefferson team developed a visualization technique to analyze one such unsolvable problem, the classic "three body problem". The difficulty in finding an analytical solution for three planets having mutual gravitational attraction is related to their interdependencies of that gravitational pull. Using a supercomputer simulation and a graphics display approach similar to that used in the trajectory model, the student was able to gain a partial understanding of the dynamics of that system.

Computational models are not fool proof either, for there are often difficulties with chaotic behavior and sensitivity to initial conditions. However, a carefully designed computational approach can often provide some insight into many "unsolvable" problems in mathematics and science.

Understanding the Trajectory Model

There are several things that the missile trajectory investigation can show. The visualization technique not only defines acceptable values for velocity and launch angle by means of a graph, but also shows that for a broad range of angles on either side of 45 degrees, the velocity required for a hit is not nearly as sensitive as it is at the extremes. When the angle approaches either zero degrees or 90 degrees, a slight change in angle requires a significant change in velocity in order to make a hit. This relationship, although intuitively obvious, is very clearly shown by the computer graphics.

Computational science models can provide alternative methods for investigating problems of all kinds. With the aid of graphics visualization, scientists can often gain additional insight into many research investigations. Sometimes an unrelated aspect of the display, such as the patterned background in the trajectory example, might be the inspiration for yet another project.

Example #2 - The Gypsy Moth

Another example we have used at Jefferson involves population dynamics of the Gypsy moth caterpillar, an extremely damaging pest of forests in the eastern United States (Milne & Milne, 1980; US Department of Agriculture, 1952). In 1869, Gypsy moths used in a scientific experiment escaped from a laboratory in Massachusetts, and since there are no natural predators here, successive generations have been ravaging forests in the northeast ever since.

Model Parameters

As with the trajectory model, there are certain equations and parameters which control the basic functioning of Gypsy moth ecology. For instance, a female moth can lay between 400 to 1000 eggs in a single season. When these eggs hatch the next spring, the tiny caterpillars crawl up to the top of forest trees and start eating leaves. There is little

damage at first, but as the caterpillars grow in size, each one can eventually consume up to three square feet of leaf surface every day. Caterpillars hatch in early spring, feed until mid summer when they develop into pupae, and after a short time, emerge from the pupae as adult moths. In the moth stage they do not eat leaves anymore but just mate, lay eggs, and die. Fortunately, there is only one generation per year.

Gypsy moths spread by several methods. Because females contain so many eggs, adult moths are too heavy to fly although they are able to crawl. Adults often lay egg masses on moveable objects such as automobiles or campers which then travel to other regions of the country. When these eggs hatch, a new infestation site becomes established.

At first, the young larva are quite small and can be blown by the wind on silk threads for distances up to ten miles. As caterpillars become larger, they can no longer travel by that method but will crawl to adjacent forest regions in search of food.

During severe infestations, trees can become completely defoliated very rapidly. Most trees will die after one or two defoliations, thus eliminating a potential food supply for the growing moth population. Caterpillars seem to like certain trees better than others; oak trees are a preferred food but the caterpillars rarely ever eat tulip poplar trees.

Gypsy moth caterpillars are not affected by many standard predators since the larva usually hide during the day and eat only at night. There are some specific predators which may eventually create a natural balance, but human intervention is an important factor in minimizing primary forest damage in eastern forests. Some human methods of control include destroying egg masses, trapping caterpillars, and spraying trees with biological or chemical controls.

As with any good scientific research, it is important to survey available literature on the subject to review what has been done previously, and to establish precise parameters for an accurate simulation. A more thorough research of Gypsy Moths is recommended before building a realistic model, but the information provided above is probably sufficient to get a prototype started.

Stages in Prototype Development—Importance of Validation

With a situation as complex as Gypsy moth ecology, it is important to develop the prototype simulation in stages, validating each new characteristic as it is added to the model. To try to put every conceivable aspect of a computer simulation into a model at the beginning and later attempt to validate the result will usually lead to serious difficulties.

A first stage of development in a Gypsy moth simulation would be to model a simple, closed population of moths with a standard birth rate and unlimited food supply. The time step could be done on a daily basis, but probably a yearly summary of Gypsy moth statistics would be sufficient since the desired outcome of this project is to show the qualitative behavior of a population over a long period of time. The expected behavior of this first stage would be to make certain that the model shows a typical exponential growth curve such as that shown in Figure #4. Gradually, other characteristics should be added, and the accuracy checked each time against expected behavior of the population. Figure #5 shows a repeated collapse of the Gypsy moth population due to overpopulation and starvation when all of the food supply is consumed.

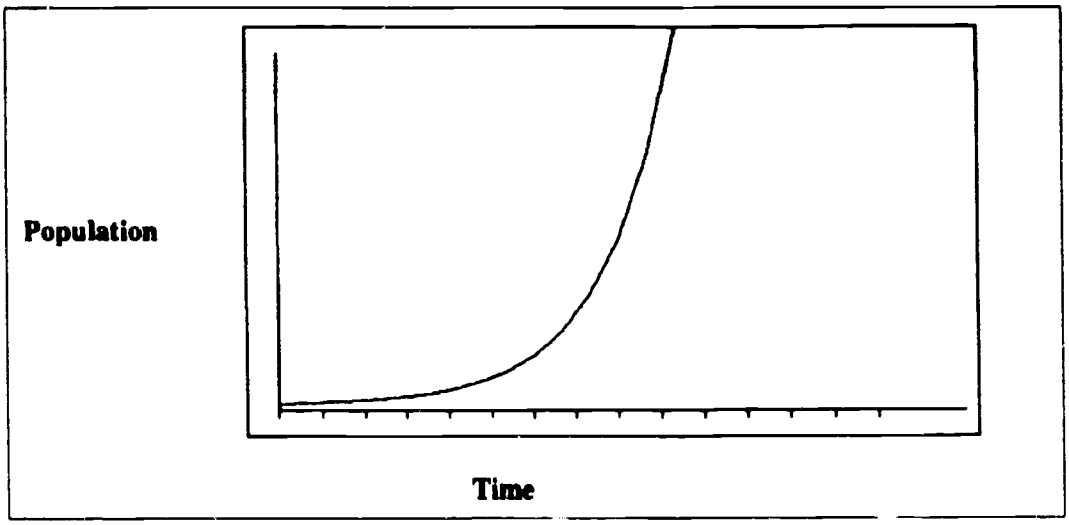


Figure 4. Typical exponential growth curve

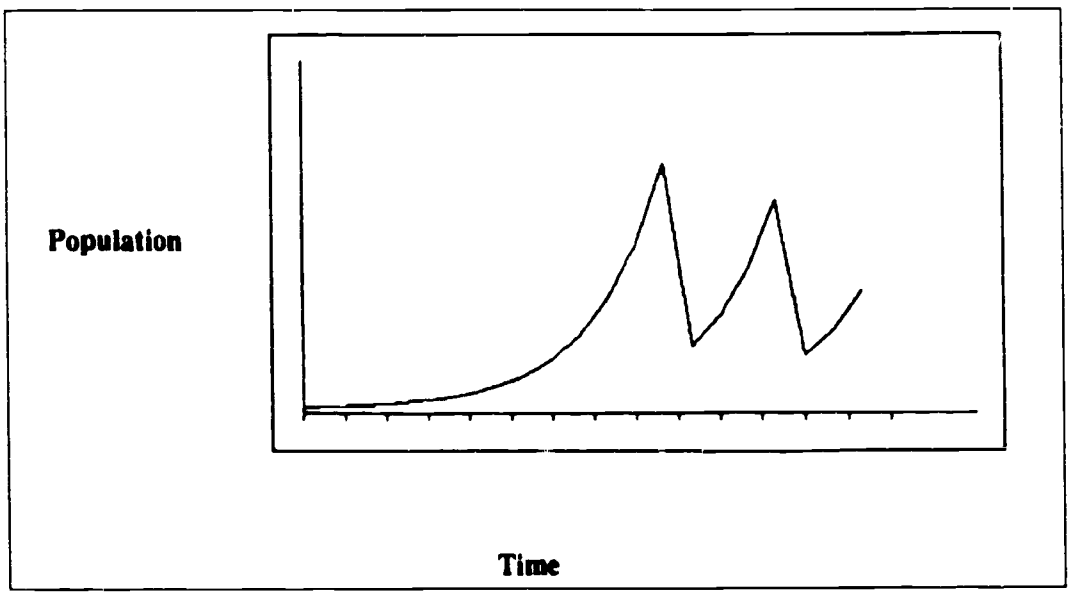


Figure 5. Growth curve showing repeated overpopulation and starvation

There are always random factors which affect such simulations, but it is important to avoid adding random occurrences until all aspects of the model have been validated. Early introduction of random numbers to make things "interesting" should definitely be avoided. It is usually very difficult to determine whether the behavior of a model is the result of the randomness, or the result of complex interactions among other factors in the simulation. Constant values within an expected range should be used initially to prove the model is working correctly. Random numbers can always be added later.

Expanding the Model

The prototype for a sample population will usually become quite involved as various aspects of a realistic simulation are included. Even so, this simulation could still be handled quite easily by a standard PC. The need for a supercomputer becomes apparent when a much larger region is modeled using many different "cells", all running in parallel. This modeling technique is often referred to as a cellular automaton (Toffoli & Margolus, 1989). Each cell would carry the fundamental rules of the initial prototype, but now migration of caterpillars and moths into adjacent cells can be included in order to look at the overall dynamics of Gypsy moths spreading into new forest areas.

It will be necessary to maintain much information about each of the cells, such as the number of caterpillars, birth rates, death rates, food supply, types of trees, history of past defoliations, the existence of predators, and spraying programs. Huge arrays of data will be required to keep track of all the important information available for a large number of cells.

This problem could now become too large for a typical supercomputer run if the same graphics visualization technique used in the trajectory model is applied here. To use every pixel on the screen as a separate cell would require extensive amounts of memory which might not be practical. A better approach in this case would be to divide the screen into small squares or other geometric shapes where each polygon stands for the status of a small segment of the overall population. With fewer cells and lower data requirements, it might be possible to accumulate information for successive years and display results as an animation over time.

Figures #6, #7, and #8 show various stages of the cellular automaton approach to the simulation. The darker the color within a cell, the greater the number of Gypsy moths in that region.

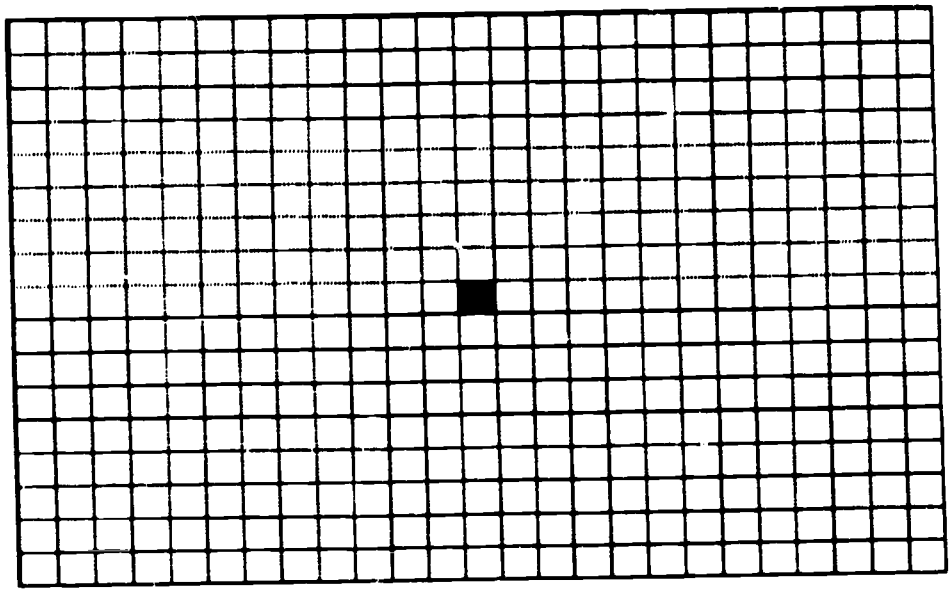


Figure 6. Initial stage of the gypsy moth population cellular automation

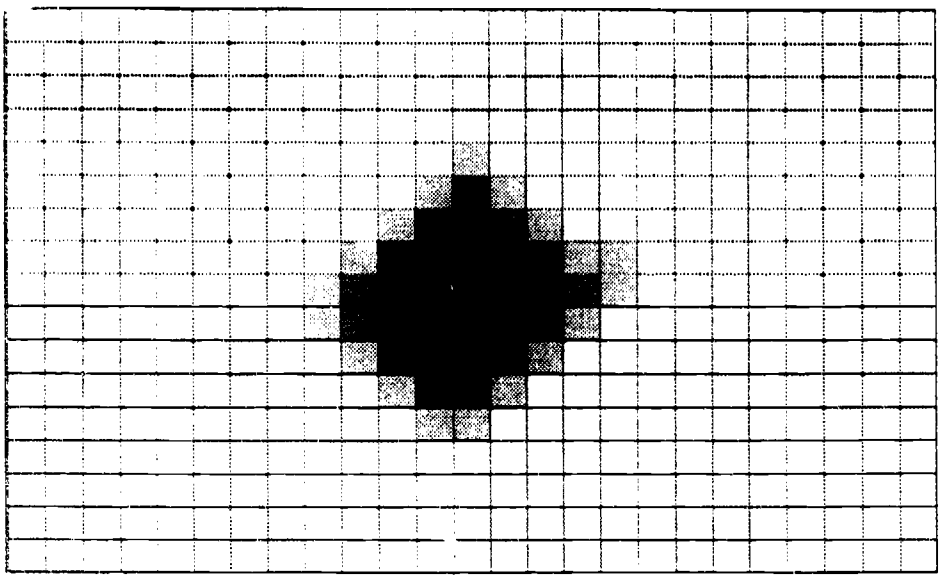


Figure 7. Intermediate stage in gypsy moth simulation

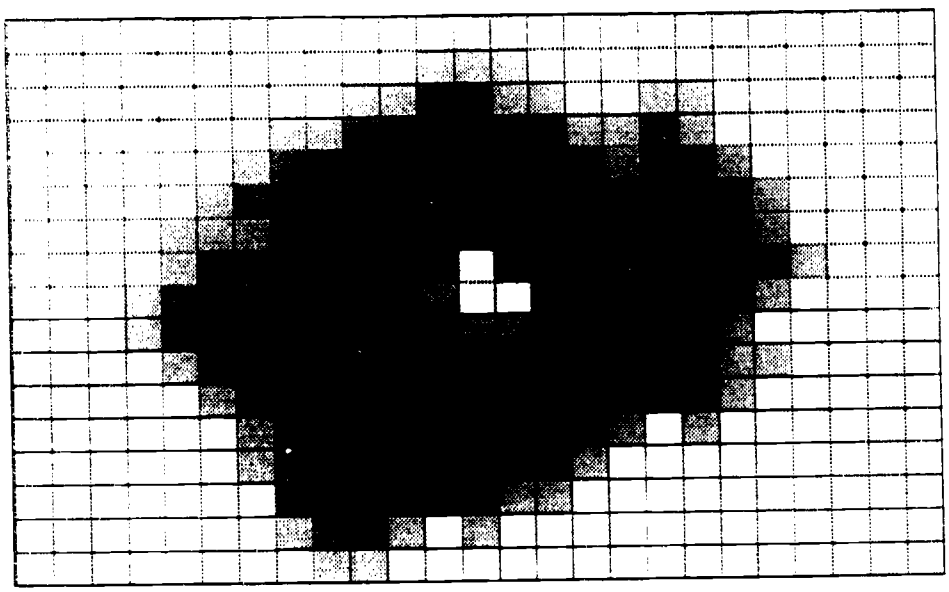


Figure 8. Final stage in gypsy moth simulation showing population collapse

Porting the Model to a Supercomputer—Conversion to FORTRAN

Although most students in high school would prefer to program in Basic, Pascal, or C, unfortunately most supercomputers prefer FORTRAN. Compared to many computer

languages, FORTRAN has a relatively simple structure and therefore compilers can optimize code to better take advantage of vectorization and parallelism in supercomputer architectures. Since FORTRAN will usually run faster than most other languages, it should be the language of choice when speed is a priority. Another advantage of FORTRAN is that it is very mathematically oriented with special features, such as exponent operators and complex number arithmetic. Also, many supercomputers already have special libraries supporting advanced mathematics and engineering routines written in FORTRAN.

Rather than rewriting every part of the program in FORTRAN, only the main algorithm and data producing routines should be converted. Most supercomputers will not directly display graphics, so the visualization portion can remain in whatever language was used to develop the prototype. The FORTRAN portion will be used to generate data files on a supercomputer. The data produced can eventually be displayed on a PC or graphics workstation to see the results.

These data files can be rather large, so additional consideration should be taken to minimize the space requirements needed to represent an image. For instance, if a full screen in VGA graphics has 307,200 pixels, then a data file of integers where each pixel position is represented by five characters (a 4-digit integer field separated by a space) will create 1,536,000 bytes of data. A file this size will not fit on most standard floppy disks! However, if each pixel can be represented by a single character and no spaces, then the file will be just the 307,200 bytes long. Even better, if it is possible to do some clever data compaction, such as packing the display data of four pixels into a single character, then the file size would be just 76,800 bytes.

Once the supercomputer portion is written in FORTRAN, and supporting graphics display routines written in any appropriate computer language, the real scientific research can begin. The trajectory model might be used to investigate basketball free throws, with and without such factors as bouncing off the backboard and including various spins on the ball.

The Gypsy moth model could be modified to explore many different scenarios. Is it possible to keep populations relatively stable by resorting only to biological controls, or must hazardous chemicals be used to save the forests? Should trees that Gypsy moths don't like be planted in regions where moths are expected to migrate in order to avoid severe defoliation? Is it possible to predict the nature of forest evolution since favorite tree species might eventually become extinct? These and many other interesting questions might be answered by a realistic computer model.

A word of warning, though. Before ever predicting the outcome of a simulation, check to see how sensitive the model is with respect to initial conditions. Sometimes a slight modification in one of the parameters can lead to totally different results. It is important to know if there is a tendency toward such chaotic and unpredictable behavior.

Finding Ideas for Supercomputer Projects

For high school students, it is sometimes disappointing to look for project ideas by researching current supercomputer applications being done at a university level. Often these investigations are very complex requiring advanced levels of mathematics, physics, and other subjects. To try to match what college professors and graduate

students are currently doing is rather ambitious and potentially frustrating for many high school students.

A better approach is to look for computational science projects in subjects where students already have considerable background. Students are often more successful applying supercomputing techniques to relationships in familiar areas of science rather than in the latest trends. There are possibilities for new and creative investigations in practically every field. After all, supercomputing power has not been available for that many years.

Current hobbies and interests are excellent sources for project ideas since students will already have a large knowledge base and such topics are typically self-motivating. It can be helpful to look for simple scientific principles which might be interrelated in some way. People may know what the basic laws of science are, but do they know how those laws interact in more complex systems?

The visualization techniques implemented in the two examples could be helpful when trying to formulate a new problem. For instance, rather than modeling a projectile, some other physical relationship could be studied using a program design similar to the trajectory example. It is important to build on the knowledge of others, but then be creative in order to discover something new. That is the essence of scientific investigation.

As an introduction to graphics, students may find the topics of fractals, chaos, and dynamics very motivational. Images in the complex plane such as the Mandelbrot set and Julia sets are very computationally intensive and good projects for introducing supercomputing techniques. The monograph by Robert L. Devaney (1989) and James Gleick's (1987) best seller, *Chaos*, are good introductions to these topics and potential sources for ideas. During the 1988 SuperQuest contest, one Jefferson student developed a research project which was inspired by Gleick's book. The student showed how the various portions of a fractal generated by Newton's method (Tuteja, 1988) were all related, and was able to describe the convergence process in a simple recursive definition. His project was judged best overall in that year's competition.

Ideas for some excellent projects can often come from simple examples. For instance, after reading about the physics of baseball, a former student and varsity athlete began a simulation of the flight of a baseball (Berkey, 1991) using an approach similar to the prototype in the trajectory example. Expanding the model to three dimensions and including many other factors, he eventually showed that a gradual slowing down of the spin on a baseball during its flight significantly modifies physical forces involved. This, in turn, affected the optimum angle with which to hit the baseball.

There are many areas in pure computer science which have interested Jefferson students, such as simulation of parallel architectures (Hargrove, 1990), multi-tasking operating systems (Rosen, 1988), and distributed processing (Brown, 1990) over networked computers. Subject areas which typically do not involve computer science have been sources for excellent projects too. One student developed a simulation of the interaction of biological oscillators in the heart (Sadananda, 1990). She showed that chaos in the heart muscle could develop under certain conditions which would result in cardiac arrhythmia. Another student used artificial intelligence techniques (Bishop, 1990) combined with raw supercomputing power to help arrange choral music in the style of famous 18th century composers such as Mozart.

Although there are many applications which are appropriate for a supercomputer to solve, there are many others which are not. For instance, interactive tutorials and

computer games are best handled by a regular microcomputer rather than a high powered supercomputer. Students will need to recognize when to use a supercomputer and how to develop parallel algorithms, since computational science will become a standard investigative technique in the next generation. Many of these technological skills can be learned in high school and will therefore be available when needed throughout college and eventual careers.

Without the help of more powerful supercomputers, many problems which scientists are facing may not be resolved. Fortunately, computer companies will continue to build faster machines, but unless there are skilled persons to use the equipment effectively, there will not be a market for those latest innovations. One thing is certain, though. Talented people with a strong background in cutting edge technologies will always be in demand.

References

Arnold, C. (1988). *What is supercomputing?* Audio Cassette and Slide Presentation. Minneapolis, MN: ETA Systems.

Devaney, R.L. (1989). *Chaos, fractals, and dynamics: Computer experiments in mathematics.* Menlo Park, CA: Addison-Wesley.

Etter, D. M. (1990). *Structured fortran for engineers and cientists* (3rd ed.). Redwood City, CA: Benjamin/Cummings.

Foley, J.D., van Dam, A., Feiner, S.K., & Hughes, J.F. (1990). *Computer graphics, principles and practice* (2nd ed.) Menlo Park, CA: Addison-Wesley .

Gleick, J. (1987). *CHAOS: Making a new science.* New York: Viking Penguin.

Karin, S., & Parker Smith, N. (1987). *The supercomputer era.* Orlando, FL: Harcourt Brace Jovanovich.

Hwang, K., & Degroot, D. (1989). *Parallel processing for supercomputers & artificial intelligence.* McGraw-Hill.

Milne, L., & M. (1980). *The Audibon Society field guide to North American insects and spiders.* Alfred A. Knopf.

Toffoli, T., & Margolus, N. (1989). *Cellular automata machines: A new environment for modeling.* Cambridge, MA: MIT Press.

Supercomputing Review. (1988). Premier Issue. San Diego, CA: London Manhattan Publishing.

The physics problem solver. (1976). New York, NY: Research and Education Association.

Insects, the yearbook of agriculture. (1952). Washington DC: United States Department of Agriculture.

NCSA realtime (1992). Video Cassette Volumes #1-#4. Urbana-Champaign, IL: National Center for Supercomputing Applications.

The International Journal of Supercomputer Applications (1990), *4*(2).

Student Research Papers

Berkey, J. (1991). *The effects of spin reduction on the launch angle that maximizes the range of a baseball.* Finalist 50th Westinghouse Science Talent Search.

Bishop, L.M. (1990). *Computer arrangement of 18th century choral music through an artificially intelligent knowledge-based system.* Semi-finalist 49th Westinghouse Science Talent Search.

Brown, D.R. (1990). *A simulation of locally optimized load balancing in distributed processing.* SuperQuest 1990 and Semi-finalist 50th Westinghouse Science Talent Search.

Hargrove, P.H. (1990). *Developing a computational model of parallel computing systems.* Semi-finalist 49th Westinghouse Science Talent Search and Winner SuperQuest 1990 Best Paper Competition.

Rosen, J.D. (1988). *The effectiveness of various priority queuing algorithms as applied to a multitasking system.* SuperQuest 1988 and Finalist 48th Westinghouse Science Talent Search.

Scheirer, E.D. (1988). *A chaotic analysis technique for the planar restricted three-body problem.* SuperQuest 1988 and Semi-finalist 48th Westinghouse Science Talent Search.

Sadananda, V. (1990). *Chaotic cardiac arrhythmias.* SuperQuest 1990 and Finalist 50th Westinghouse Science Talent Search.

Tuteja, M.K. (1988). *Paths of convergence to the roots of unity by the Newton-Raphson method: An investigation.* SuperQuest 1988 Best Paper and Semi-finalist 48th Westinghouse Science Talent Search.

Source Code - Trajectory Model

Pascal Source Code for the Full Prototype Model

```
PROGRAM Trajectory(INPUT,OUTPUT);
(* This is a prototype model for a supercomputer application.          *)
(* The program will run through a basic simulation for each            *)
(* pixel position on the screen.  Every pixel represents a             *)
(* combination of parameters  for velocity and launch angle,           *)
(* and its color will be determined by whether the missile was         *)
(* close to the target or not.  The resulting screen display           *)
(* is a way of showing the solution using a graphics                   *)
(* visualization technique.                                            *)

USES
  Graph,Crt;  (* Graph Unit allows for Turbo Graphics Access           *)

CONST xwidth = 639;   (* Maximum Horizontal pixel position             *)
      ywidth = 479;   (* Maximum Vertical pixel position               *)

PROCEDURE DrawAxes(vmax:REAL);
     (* Draw Axes on the screen and display                            *)
    (* range for launch angle and velocity.                           *)
VAR
  s:STRING;
 (* Temporary string for use in printing values in graphics           *)
BEGIN
  Setcolor(15);           (* Set color to draw axes                   *)
  Line(0,0,0,460);        (* Vertical Line  -> y-axis                 *)
  Line(0,460,639,460);    (* Horizontal Line -> x-axis                *)
  OutTextXY(0,469,'0.0');(* Print minimum velocity on screen          *)
  Str(vmax:5:3,s);        (* Convert max velocity to a string         *)
```

```
 OutTextXY(600,469,s);          (* Print max velocity on screen  *)
 OutTextXY(0,200,'Theta');      (* Label vertical axis           *)
 OutTextXY(270,467,'Velocity'); (* Label horizontal axis         *)
 outTextxy(0,0,'90.0');         (* Print maximum angle on screen *)
 OutTextXY(0,460,'0.0');        (* Print minimum angle on screen *)
END;


PROCEDURE  SimulateProjectile(velocity,theta,timestep,
             distance,range:REAL; VAR color:WORD);
(* This is the basic module for the simulation.  The laws of      *)
(* physics are used to determine the position of a projectile     *)
(* over time, given the initial velocity, launch angle (theta)    *)
(* timestep between different time values, the distance to the    *)
(* target and how close the projectile can come and be counted    *)
(* as a hit. The values for all of these parameters are passed    *)
(* to this procedure from the primary routine.  The procedure     *)
(* will return a color value which indicates whether the          *)
(* projectile hit or missed the target.                           *)


 VAR  vx,vy, (* Vertical and horizontal components of velocity   *)
        time,            (* Current time for basic simulation    *)
        x,y:REAL;        (* Current horizontal and vertical positions*)

 BEGIN
   vy := velocity * sin(theta); (* Find y-comp. of velocity       *)
   vx := velocity * cos(theta); (* Find x-comp. of velocity       *)
   time := 0;                   (* Reset time to zero             *)
   (* Basic prototype loop *)
   REPEAT              (* Follow Parabolic path of projectile    *)
     time := time + timestep;       (* Increment timestep        *)
     y := vy * time - 16 * time * time;  (* Calculate height      *)
   UNTIL y <= 0; (* Continue until projectile hits the ground    *)
   x := ABS(vx * time - distance); (* How close to target        *)

   IF x < range               (* If projectile within range     *)
     THEN                      (* Considered a hit              *)
      color := 15 (* color for a "hit"                          *)

     ELSE                      (* Else it's a miss              *)
       color := (trunc(x/range) mod 2); (* Alt. colors          *)
                               (* for various misses            *)
 END;


 PROCEDURE DoScreenPositions(distance,vmax,timestep,range:REAL);
 (* This section will step through every pixel position on      *)
```

```
(* the screen and send the appropriate values to the module     *)
(* "Simulate". It will then plot the appropriate shade on       *)
(* the screen depending upon whether the result was a hit or    *)
(* a miss.                                                      *)
(* Parameters: dist. to target, maximum velocity, timestep      *)
(* for simulation, range considered close enough for a hit.     *)

VAR vertical,horizontal:INTEGER; (* Pixel screen coordinates    *)

  theta,thetastep,(* Angle values & dist. between choices       *)
  velocity,vstep, (* Velocity values & step between choices     *)
  x,y :REAL;      (* Vertical & horizontal position in feet     *)
  color:WORD;     (* Color of pixel that will be plotted        *)

BEGIN
 thetastep := Pi/((ywidth-19)*2);
          (* Compute distances between successive angles       *)
          (* Pi/2 divided among vertical pixel positions       *)
 vstep := vmax/xwidth;
     (* Compute distance between successive velocities         *)
     (* Max velocity divided by horizontal pixel positions     *)

 theta := 0;                        (*  Set angle to zero       *)

 DrawAxes(vmax);                    (* Draw Axes for reference  *)

 FOR vertical := 1 TO ywidth-19 DO  (* Do all rows of pixels    *)
  BEGIN                             (* Start next row of pixels *)
   velocity := 0;           (*  Reset velocity for next row    *)
   FOR horizontal := 1 TO xwidth DO (* Do all pixels in row     *)
    BEGIN
      SimulateProjectile(velocity,theta,timestep,
                         distance,range,color);
(* Send current set of parameters to basic simulation routine  *)
      PutPixel(horizontal,ywidth-19-vertical,color);
        (* Color the current pixel position on the screen      *)
       velocity := velocity + vstep;
        (* Try next velocity combination *)

     END; (* End of row of pixels *)
    theta := theta + thetastep;
      (* Go to next angle by adding increment *)

  END  (* Continue until all rows are done *)

END;  (* End of procedure *)
```

```
PROCEDURE Main;

VAR Gd,Gm:INTEGER; (* Gd - Graphics Device, Gm - Graph Mode      *)
   vmax,           (* Max. velocity to try in the simulation   *)
   distance,       (* Distance to the target                   *)
   timestep,       (* Timestep for the simulation              *)
   range:REAL;     (* Radius from target which counts as a hit *)

BEGIN
 writeln('Enter distance to the target:');
 readln(distance);
 writeln('Enter maximum velocity:');
 readln(vmax);
 writeln('Enter time step for the simulation:');
 readln(timestep);
 writeln('Enter distance from target which will count as a hit:');
 readln(range);
 Gd:=Detect;      (* PC will determine what kind of graphics  *)
 InitGraph (GD, GM, 'C:\TP');  (* Initialize graphics window  *)
                               (* Path to BGI drivers in C:\TP *)
 DoScreenPositions(distance,vmax,timestep,range);
    (* Call main routine *)
 Drawaxes(vmax);               (* Redraw Axes  *)
END;

BEGIN
 main
END.
```

FORTRAN Source Code for Generating Supercomputer Data

```
      PROGRAM SMODEL
C     FORTRAN program to generate screen data for the trajectory
C     model. This program will be run on a supercomputer. The
C     data file generated will be transferred to a PC where the
C     results can be displayed graphically using Turbo Pascal.
      PARAMETER (PI=3.1415962)
C
      REAL V,VMAX,VSTEP,VX,VY,THETA,THSTP,X,Y,T,TMSTP,DIST,RANGE
C     Real Variables: current velocity, maximum velocity,
C       velocity step, x-component of velocity, y-component,
C       angle (theta), theta step, x-coordinate of projectile,
C       y-coordinate, distance, and range
C
```

```
      INTEGER COLOR,ROWMAX,COLMAX,ROW,COL,TEMP,COUNT,POS
C     Integer Variables:  pixel color, maximum row value, maximum
C       column, row position, column position, temporary value
C       for packing screen information, count of packed values in
C       a character, position in the character array.
C
      CHARACTER*160 LINE
C     Character Variable:  Line which holds 160 characters, each
C       of which holds information for four pixels.  Total width
C       is 640 pixels
C
      OPEN(9,FILE='STATS.TXT')
      OPEN(10,FILE='scr.dat')
C     Open Files for input and output
C
800   FORMAT(I4,I4,F10.3,F10.3,F8.2,F8.2)
      READ(9,800,END=111) ROWMAX,COLMAX,VMAX,DIST,TMSTP,RANGE
C     Read Current statistics for run: ROWMAX=480, COLMAX=640,
C       VMAX=1000.0, DIST=600.0, TMSTP=0.1, RANGE=20.0
        WRITE(10,800) ROWMAX,COLMAX,VMAX,DIST,TMSTP,RANGE
C     Write header to data file for later use by display program
C
C     Initialize Parameters
      VSTEP = VMAX/COLMAX
      THSTP = (PI/2)/ ROWMAX
      THETA=0
      COUNT = 1
      TEMP = 0
C
C     Outer Loop ->  Do all rows of Pixels
      DO 300 ROW=1,ROWMAX
      POS = 1
      V=0
      THETA=THETA + THSTP
C
C     Inner Loop -> Do one row of pixels. Data will be compacted.
      DO 310 COL=1,COLMAX
C     Initialize remaining variables for specific run
      V=V + VSTEP
      T=0
      VX=V * COS(THETA)
      VY=V * SIN(THETA)
C
C     Simulated REPEAT/UNTIL loop which is basis of simulation
320   T=T+TMSTP
      Y = VY * T - 16 * (T ** 2.0)
      X = VX * T
      IF (Y.GT.0) GOTO 320
```

```
C     End of REPEAT/UNTIL loop structure
C
C     Check to see if projectile came close to target
C     Color values will be a 1 (hit), or else a 2 or 3 (misses)
      IF (ABS(X-DIST).LE.RANGE) THEN
        COLOR=1
      ELSE
        COLOR = X/RANGE
        COLOR =  MOD(COLOR,2) + 2
      ENDIF
C     Pack information of four runs into one character
      TEMP = TEMP*4 + COLOR
      COUNT = COUNT + 1
C     If character has four data points, put value into buffer
C     and continue.  Note that input and output are usually the
C     slowest aspects of computer operations.  This is an attempt
C     to minimize constant writes to a file for each separate
C     character by writing a block of characters at once. Even so
C     it may be necessary to remove the compaction operation from
C     within the inner loop since the data dependency might make
C     it difficult for a supercomputer to do the heaviest
C     computation in parallel.
      IF (COUNT.GT.4) THEN
        LINE(POS:POS) = CHAR(TEMP)
        POS = POS + 1
        COUNT = 1
        TEMP = 0
      ENDIF
310   CONTINUE
C     End of Inner Loop
C     Write out the buffer
      WRITE(10,850)LINE
850   FORMAT(A160)
300   CONTINUE
C     End of outer loop
C     Close all files and terminate the program
111   CLOSE(9)
      CLOSE(10)
      STOP
      END
```

Pascal Source Code for Displaying Supercomputer Data

```
PROGRAM ImageRestore(INPUT,OUTPUT);
(* This program reads data generated by a supercomputer and  *)
(* then displays the information graphically on the screen.  *)
(* The data been encrypted in order to conserve space. Each  *)
```

```
(* character in the file holds the information for four          *)
(* values.  Those values are regenerated by using the           *)
(* operations DIV and MOD on the ORD of the character and        *)
(* then plotting the related pixels in the proper place on       *)
(* the screen.                                                   *)

USES
  Graph,Crt; (* Graph Unit allows for Turbo Graphics Access      *)

VAR infile: TEXT;          (* File containing screen data        *)

PROCEDURE DisplayData(filename:string);
(* This procedure opens the appropriate file and displays       *)
(* the data on the screen after extracting four values from     *)
(* each character in the file.                                  *)

VAR group,          (* Groups of four pixels forming a row      *)
    count,            (* Which pixel within each group          *)
    val,   (* Integer representing character read from file     *)
    xwidth,ywidth,   (* Screen dimensions from input header     *)
    vertical,horizontal:INTEGER;
                                  (* Pixel coordinates on screen *)
    color:WORD;      (* Color of pixel that will be plotted     *)
    ch: char;        (* Character to be read from the file      *)
    vmax,thstep,dist,range:REAL;
                     (* Dummy Variables in header               *)

BEGIN
  Assign(infile, filename);  (* Open infile to access data      *)
  Reset(infile);
  Read(infile,ywidth,xwidth,vmax,thstep,dist,range);
  FOR vertical := 0 TO ywidth DO  (* Do all rows of pixels      *)
    BEGIN                    (* Start next row of pixels       *)
      horizontal := 3;  (* Initialize horizontal position      *)
                   (* for first four pixels                     *)
      FOR group := 0 to 160 do      (* Do a row of pixels       *)
       BEGIN
         read(infile,ch); (* Get character from the file       *)
         val := ORD(ch);         (* Convert to an integer      *)

         FOR count := 0 TO 3 DO   (* Extract four pixels       *)
          BEGIN
            color := val MOD 4;  (* Value for next pixel       *)
              (* Change to the appropriate screen color       *)
            if color = 1 then color := 14
            else color := color *2;
            putpixel(horizontal-count,470-vertical,color);
              (* Plot pixel in the appropriate position       *)
```

```
        val := val DIV 4;    (* Get next data value              *)
       END;                  (* End of FOR loop          *)

      horizontal :=horizontal + 4;
         (* go to next four pixels  *)
    END; (* End of one row of pixels   *)
  END   (* Continue until all rows are done *)
END;  (* End of procedure *)

PROCEDURE Main;

VAR Gd,Gm:INTEGER; (* Gd - Graphics Device, Gm - Graph Mode *)
  filename:string;    (* Name of supercomputer data file *)
BEGIN
  WriteIn('Enter filename containing supercomputer data');
  ReadIn(filename);
  Gd:=Detect;  (* PC will determine what kind of graphics    *)
  InitGraph (GD, GM, 'C:\TP');
                  (* Initialize graphics window              *)
                  (* Path to BGI drivers in C:\TP            *)

  DisplayData(filename);          (* Call main routine       *)
END;

BEGIN
 main
END.
```

Contacts for More Information

Donald W. Hyatt, Computer Systems Laboratory Director
Thomas Jefferson High School for Science and Technology
6560 Braddock Road
Alexandria, Virginia 22312
(703)-750-8300 main office (703)-750-5026 computer lab

SuperQuest
Cornell Theory Center
424 Engineering and Theory Center Building
Ithaca, New York 14853
(607)-255-4859

Chapter 8

Scientific Visualization in Chemistry, Better Living Through Chemistry, Better Chemistry Through Pictures: Scientific Visualization for Secondary Chemistry Students

ROBERT R. GOTWALS, JR.

Like it or not, visual literacy was ordained the moment the first television flickered on a half century ago, and no country has adapted better than ours. With information from every conceivable source coming at us more and more rapidly each day, the old ways of processing it have become inadequate. Because images pack countless insights and ideas into a fleeting moment, and communicate information more efficiently than words, they have become the preeminent means of relaying knowledge.

So says Leonard Steinhorn (1992) in his editorial entitled "Whiz Kids in America." He suggests that American youth, in the many hours devoted to MTV and Nintendo, are on the cutting edge of the Information Age. He suggests that "through video and computer games and all the fast-paced and disjointed videos on MTV, young Americans have been processing information in a way that makes little sense to the uninitiated, but is really the wave of the future." He also states that "computer technology has already made more knowledge available at our fingertips than our parents had at their local libraries, and products soon to be mass marketed will make what we have now look like a children's book." He also states that "the problem is how to digest such immense amounts of information without being completely overwhelmed."

Chemistry is a hard subject to teach and a hard subject to master. Many of the concepts and ideas we try to teach do not lend themselves to students' natural intuition and knowledge-base. Most of the concepts are highly abstract and can only be described using complex mathematics. T.C. O'Haver (1992) describes the problem as being one of macroscopic vs. microscopic and discrete vs. continuous. Chemical reactions are constantly in flux, a multitude of indivdual molecules gyrating about, bumping into other molecules and changing. We can observe these changes on a macroscopic level;

modern atomic theory has allowed us to interpret these observations using the microscopic paradigm. But it is very difficult to help the fledgling chemistry student understand the connection between the observable and the reality. We frequently use discrete mathematical expressions to help our students try to quantify very continuous phenomenon. How many of us teach PV=nRT and equilibrium constants? Very important and basic concepts, no doubt. As mathematical models go, these equations are probably not bad approximations of reality, especially for the beginner. These equations are, however, static. They define the chemical environment for only a fraction of a moment. In other words, what are we doing to help our students always remember the fluid, dynamic, continuous nature of chemical reactions?

Probably a better question is: what tools do we have at our disposal to make sure kids see the macro and the micro, the discrete and the continuous? We certainly have the overhead and the chalkboard. We can write down equations and balance them. We can do labs, try to help the students see the connection between the mathematical model and the reaction. We can build models using ball-and-stick kits, trying to help our students see the three-dimensionality of molecules. With the advent of the video disk, students can now watch simulations of experiments conducted by others. All very useful tools, indeed. What is missing is a tool that allows the students to collect data that more closely approximates the dynamical nature of chemical systems and then allows the students to evaluate that data in a manner that allows them to "see" inside that system. In order to really understand the workings of a chemical system, students need to be able to manipulate the conditions or variables of that system, then interpret how that manipulation changes the dynamics of the reaction.

The tool—the use of computational science coupled with scientific visualization—has been around for a number of years. Computational science can be defined as the use of numerical and computer techniques to solve complex scientific problems. Numerical techniques include the use of well-defined algorithms such as Newton's method for finding the roots of an equation or the Runge-Kutta algorithm for solving ordinary differential equations. Computer techniques include ideas such as the use of iteration, recursion, or multiple processors to perform "number-crunching" calculations that are difficult or impossible to do by hand-held calculators. Computational science offers chemistry teachers and students with the capability to go beyond simple closed analytical equations such as PV=nRT and into more complex equations such as van der Waals equation of state for a gas, which are difficult to solve analytically. Many of the numerical and computational techniques that are useful in chemistry require only that the teacher/student have a background in algebra. Many interesting chemistry problems can be solved using the computational power found in most spreadsheets or with some simple programming using BASIC or Pascal.

Why do we not use this tool? The bottom line is that computational science and scientific visualization was not a topic typically taught to future chemistry teachers in undergraduate chemistry programs; indeed. it has not become a central part of science curriculums even now. A big reason for this is the rapidly changing nature of computer technology. ETA Systems, in a slide presentation on supercomputing, uses an interesting analogy in describing the amazing growth of computer technology over the past 20 years. It compares the technology of building airplanes to the technology of building computers. The presentation suggests that if airplane technology had increased as rapidly as that of computers, it would now be possible to build an aircraft that could fly six million miles on 28 gallons with a passenger load of 100,000 people at a cost of $12.50

per person. The implication is that what was possible only to a few chemists five years ago using the most expensive of supercomputers is now possible to the masses using desktop personal computers. The unfortunate part of this is that the education and the training to use this technology has not undergone the exponential growth that the development of these new technologies has. The educational component is only functioning well at the graduate school and post-doc level, while lagging years behind at the graduate level and decades behind at the secondary level. The purpose of this chapter, and this entire monograph, is to look at some of these new technologies and how they might be implemented at the secondary level. This specific chapter—scientific visualization in chemistry—will discuss the hows and whys of scientific visualization, with a review of platforms (computer hardware and software) capable of supporting scientific visualization.

Scientific visualization is a term that can be defined many ways. Generally speaking, it is the use of computer graphics to process numerical data into two-dimensional and three-dimensional visual images. The difficulties inherent in attempting to reveal patterns hidden in reams of numerical data is obvious. Visualization allows the scientist to translate data into pictures, with the hope that patterns, inconsistencies, and new questions will be generated by the images. The National Science Foundation, in its publication "Visualization in Scientific Computing" (1987) and printed in a special issue by the Association of Computing Machinery, states that visualization is a form of communication that transcends application and technological boundaries. The report suggests that visualization is a tool for discovery, understanding, communication, and teaching. The monograph defines visualization as being a "tool for both interpreting image data fed into a computer, and for generating images from complex multi-dimensional data-sets." The monograph claims that "an estimated 50 percent of the brain's neurons are associated with vision" and that "visualization in scientific computing aims to put that neurological machinery to work". It claims that in modern science, the fundamental problem is the *information-without-interpretation* dilemma. In support of this claim, the issue states that "today's data sources are such *fire hoses* of information that all we can do is gather and warehouse the numbers they generate." Typical high-volume data sources include such things as supercomputers and medical imaging technologies such as CAT scans and MRIs.

In the field of chemistry, the advent of supercomputing technologies have allowed chemists to delve deeper into the inner mysteries of atomic structure. Mathematical equations such as the Schroedinger equation, once unsolvable by chemists, can now be approximated using high-performance computers and complex numerical techniques. Indeed, chemists have traditionally been the leaders in the use of computational science and high-performance computers for solving grand questions. A number of research chemists were interviewed for this chapter; without exception, all of them stressed the importance of computational science as a fundamental tool for the research chemist. None of them argued for the replacement of the wet lab by computers, but none argued for the use of the wet lab without computers. The improvements in the field of computer-aided chemistry have enabled chemists to design and study structures and reaction mechanisms without having to master computers or quantum chemistry. These improvements in ease of use make the feasibility of the use of these tools for beginning-level students much higher.

Hirschy (1990) states that computational chemistry packages "now mimic a chemist's usual way of conceptualizing data—by visualizing it. Chemists traditionally

explore ideas by sketching pictures and constructing models to communicate information about molecular structure, orientation, and reaction pathways." She also states that in designing a synthetic route to a desired compound, "seeing a molecule's electronic structure is an essential part of what enables chemists to predict how a substitution will affect a molecule's geometry and its underlying electronic structure." These packages also have the capability of allowing chemistry teachers and students to build molecules out of individual atoms or molecular fragments, manipulate them, and calculate molecular properties such as electron densities, molecular orbitals, ionization potentials, and electrostatic potentials.

Getting Started in Scientific Visualization in Chemistry

As mentioned in the Introduction, scientific visualization in chemistry is a relatively new topic even in the professional community. Little work has been done in trying to incorporate these technologies into the secondary science classroom. One of the hopes of this series is that it will serve to entice, motivate, excite, and otherwise encourage secondary science teachers to consider the use of new technologies in their teaching. From the perspective of this author, a classically-trained chemist with no background in computational chemistry, visualization or formal training in computer science, participation in a nationwide contest has revealed a potential gold mine of opportunities for improving the teaching of chemistry students at the secondary level.

The SuperQuest Supercomputing Challenge

The SuperQuest Supercomputing Challenge is a national scientific problem-solving competition sponsored by the National Science Foundation, the IBM Corporation, and the Cornell National Supercomputer Facility (CNSF). The purpose of this contest is to provide students with the opportunity to use high-performance supercomputers to solve relevant and interesting science problems. Students form teams of three or four students with a teacher-coach and develop individual or team proposals. Teams who score the highest are invited to spend three weeks at one of the national supercomputing facilities learning about computational science, high-performance computers, and visualization of data. Students then return to their home schools with state-of-the art scientific workstation computers (complete with graphics terminals), access to the national electronic research network (Internet) and access to the supercomputer facility for one year.

During the academic year, students finish their projects and submit the results in hopes of winning scholarship awards. Blair High School's experiences and successes in the SuperQuest contest have changed the way we think about teaching science. Our curriculum has been modified to include formal courses in Computational Methods, Modeling and Simulation, and Mathematical Physics. Other courses, including introductory chemistry, have been modified to make use of the substantial data and computational resources available from the Internet and from the supercomputing facilities. Our computer science hardware has increased dramatically as a result of winning teams over a three-year period. We now have several multiuser machines in the building complete with sophisticated graphics capabilities; these facilities rival those found at many colleges, universities and research institutions. The outlay of funds

for this equipment has been virtually zero, since all the equipment is awarded permanently to the school for SuperQuest finalist teams. Coupled with the machines found in our regular computer lab (Macintoshes and IBMs), we have the computational facilities at our disposal to allow the students to investigate interesting problems.

Several short examples of possible research are illustrative. One student is doing a three-dimensional, time-driven simulation of dissolved oxygen levels in a section of the Potomac River. Her simulation investigates the changes that occur at various levels and places in the river following discharge from an upstream pollution source. Once the simulation and the data are complete, the results will be compared with actual data collected from the literature. The mathematical equation being used to run the simulation is a relatively simple algebraic calculation that can be easily solved by hand for a single place in the river at a specific instance. The student researcher is able to make use of the number-crunching capabilities of the computer to generate a large amount of data for multiple places over an extended period of time. A graphical visualization of the data over time will be the final part of this research. This visualization will show changing values in dissolved oxygen in a three-dimensional, color-coded fashion. Another student is looking at the formation of soap bubbles and their geometry. Using complex mathematics, he has been able to develop a visualization of the formation of soap bubbles that allows him to introduce changes in the input parameters to investigate the surface geometries of different bubble formations. The computer model generates the data; the visualization software allows him to look at the data results, communicate those results, and develop more interesting questions to be investigated. While this student has the mathematical abilities to create such a model, the success of his research comes primarily from access to the newest of technologies, coupled with the drive and desire to learn and utilize these technologies.

Incorporating Computational Science and Scientific Visualization into the Chemistry Classroom

There are several key issues that must be addressed by the chemistry teacher interested in using visualization techniques for chemistry students:

1. What is the concept that I want to teach? Does the teaching of that concept lend itself to a multi-dimensional portrayal (such as electron orbitals)? Do I want to be able to show the dynamic changes that occur over time or over the course of some event, such as the effect of volume changes under increasing temp or pressure?
2. What are my data sources? Can the data be generated from a laboratory experiment, spreadsheet, computer program, or commercial software product? Are there opportunities for collecting large amounts of data from long-term experiments or from multiple groups? Are there data sources from the literature or computer databases that would be worth investigating (such as the Brookhaven Protein Data Bank, a commercial database of protein structures)?
3. What computational platforms are available? Possible platforms include personal computers such as IBM PS/2's or Macintosh computers, larger machines such as a VAX minicomputer or an IBM RS-6000 workstation with graphics terminals, or access to a supercomputing center. As mentioned previously, one of the most exciting developments of recent times is the availability of sophisticated software support for personal desktop computers. Specific examples will be discussed later in the chapter.

4. To what extent do I want the student to be able to manipulate the conditions of the visualization? Options include a range from no manipulation ("canned" software) to complete control of data being visualized.

One of the hopes of this author is that science teachers, chemistry teachers in particular, will begin to develop materials that make use of newer computer technologies for use in the classroom, and begin to share these materials throught the traditional professional channels such as publication and presentation at national scientific conferences. Based on interviews with practicing research chemists, participation in science contests with students, and a thorough reading of the literature, this author is convinced that the tools and technology are accessible and understandable to high school chemistry students, and that they will generate a much better understanding of the nature of chemical reactions and to generate substantial excitement about chemistry and chemical research.

Learning How to Play the Game: A Baseball Analogy

Many of us have played baseball or other organized sport at some time in our lives. As young children, we knew and understood the steps from Little League to the varsity team at our high school, on to college baseball. Then, if we were good enough, a chance to play minor league ball, a stepping stone to the major leagues. Hopefully, if our team was good enough, we might win the World Series! The idea was that to play in the big leagues, we had to start at the bottom with the fundamentals, moving steadily to more sophisticated skills and techniques. The analogy to the use of computational science and scientific visualization in science and mathematics teaching is identical. Our goal as teachers is to prepare our students to succeed in the big leagues, the world of scientific research. Some of our students will win the "World Series" by discovering some significant new finding or by winning a Nobel Prize. But the path starts with us; we teach the basics, and hopefully help our students move on to higher levels of scientific research and problem-solving. It is important for us to realize that the fundamental skills in demand now and in the next 50 years are changing; the use of computational technologies has become as important as being able to titrate an acid. The baseball analogy also applies to teachers.

One of the exciting things about the use of visualization in chemistry teaching is the newness of it, a sort of sense of breaking new ground. We, as well as our students, need to begin with the basics, slowly working our way up to higher levels of sophistication. It's a journey that we—teachers and students—can take together, all the while modeling the learning process for our students. In the following section, a variety of tools for scientific visualization are described. Needless to say, it is difficult and not necessary to describe all of the products available on the market today. A compendium of software products available is included in the Appendix.

Little League: The Basics

We've all done scientific visualization; we've just never called it that. The process of graphing data on an x-y coordinate system is an example of scientific visualization in its most basic form. The concept here and with the most sophisticated systems is

basically the same: to take numerical data and be able to determine patterns, make generalizations, and generate more questions. Data that follows a linear growth pattern represents very different data from nonlinear data.

Many science teachers require their students to generate graphs, be it by hand or by the use of a computer software package. Modern spreadsheets, available on personal computers, not only have the ability to generate data easily and to perform sophisticated calculations on data, but also are increasingly impressive in their graphical representation of data. Software such as Microsoft Excel and Cricket Graph can generate high-quality two-dimensional graphs of data. Coupled with built-in functions, most spreadsheets can be useful in performing some of the tedious arithmatic/computational work. Data can then be plotted or "visualized" using the built-in graphics generator. Consider a discussion of bonding and antibonding orbitals when discussing atomic structure. Using a technique from computational chemistry known as linear combination of atomic orbitals (LCAO), it is possible to numerically approximate the wavefunction for a diatomic molecule. Figure 1 shows a simple spreadsheet used to generate the data. Figure 2 shows a simple two-dimensional plot of the data. Students can make predictions about the internuclear distance and other variables affecting the system. Generation of graphical representations of that data makes interpretation easier and more revealing.

LINEAR COMBINATION OF ATOMIC ORBITALS FOR HYDROGEN

0.529 =BOHR RADIUS
0.7071 =1/SQRT 2
1.4664 =1/SQRT (A^3*PI)

Hydrogen Molecular Orbitals

Internuclear Dist.= 0.8

ATOMIC ORBITALS

Radial From A	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4
Radial From B	-2.4	-2.2	-2	-1.8	-1.6	-1.4	-1.2
Psi Is A	0.071	0.104	0.152	0.221	0.323	0.472	0.6888
Psi Is B	0.016	0.023	0.033	0.049	0.071	0.104	0.152
Bonding	0.061	0.061	0.131	0.191	0.279	0.407	0.594
Antibonding	0.039	0.039	0.084	0.122	0.178	0.260	0.380

Figure 1. A spreadsheet of Linear Combination of Atomic Orbitals (LCAO) for hydrogen

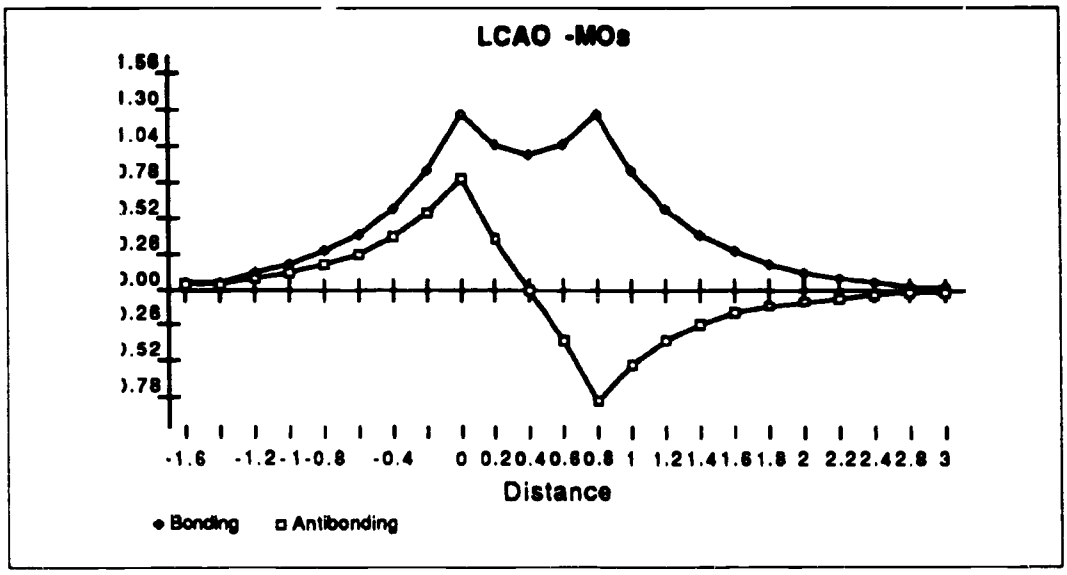


Figure 2. Graph of hydrogen bonding, showing bonding and antibonding orbitals

A second beginning place for teachers and students is through the use of controlled simulation software. These are public domain (free) and commercial software products that present the student with a menu of simulations or animations. The advantage of these products is that they are easy to use, for both student and teacher. The disadvantage is that they usually offer limited opportunity (if any) for the student to modify the situation to generate answers to "what-if" questions. If the concept of interest is not one of the menu choices, then the student has to use other means to answer that question. An example of this type of software is Organic Reaction Mechanisms, published by Falcon Software. Designed to teach organic chemistry, the software presents the student with a fairly complete list of typical organic reactions, such as addition, substraction, halogenation, Diels-Alder, and others. The student can look at examples of these reactions, such as the bromination of an alkene. The software presents the student with an overall reaction mechanism, then with the opportunity to run an animation of the reaction. During the animation, which can be run at three different speeds, the program shows parts of the reactants leaving, changing polarity, and combining with other fragments to form products. A detailed textual description of what is occuring accompanies the animation. The student can stop and re-start the animation at any time. Some of the reactions are accompanied by an inset window that graphs the reaction energetics in a "real-time" fashion while the animation is happening. Figure 3 shows a frame from the halogenation of benzene. Notice the reaction energetics graph on the right of the screen.

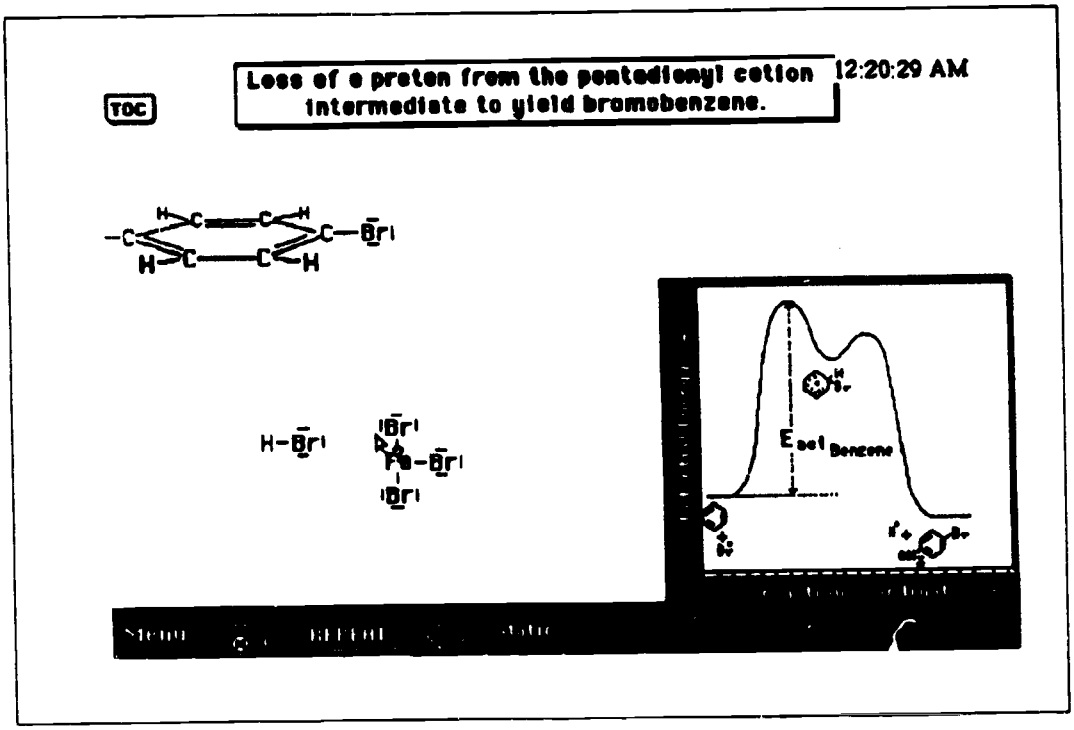


Figure 3. Screen shot of an animation showing the halogenation of benzene

This type of software demonstrates a much more effective mechanism for helping the student to understand reaction mechanics than the traditional pencil-and-paper drawing or trying to build ball-and-stick models. This particular software also shows the changes in the stereochemistry as the reagents begin to react. It is a simple program, but a substantial improvement over more commonly used methods. Beginning attempts at Blair to use this particular software in teaching beginning organic to introductory chemistry students has been rewarding, both for teacher and student.

A third starting place for scientific visualization is with the use of "limited-capability" software. This is software that allows the student to input information about a chemical system and have the software generate some information about that system. Many of these programs are inexpensive or free, and run on a variety of platforms. An example of this type of software is Molecular Editor, for the Macintosh computer. This software presents the student with a "palette" of common atoms, bonds, and other structures. The student can build a graphical model of simple or complex molecules by connecting the appropriate parts, much the same as he or she might do with molecular model kits. As with the kits, the student can rotate the molecule three-dimensionally around any of the three axes. Additionally, the student can query the software to generate an information grid about the molecule, showing bond angles, torsion angles, and other types of molecular information. Figure 4 shows a model of a protein, using the ball-and- stick representation (wire-frame and space-filling representations are other options). The insert window shows a typical drawing palette that the user sees. The user

can select any of the items from the palette for use in generating a visualization of a molecule. The program will also generate stereoscopic images of the generated model, if you are able to focus the images into a stereo view (I have never been able to do this!). Software of this type is not only fun to use, but it is informative to the student while being easy to master.

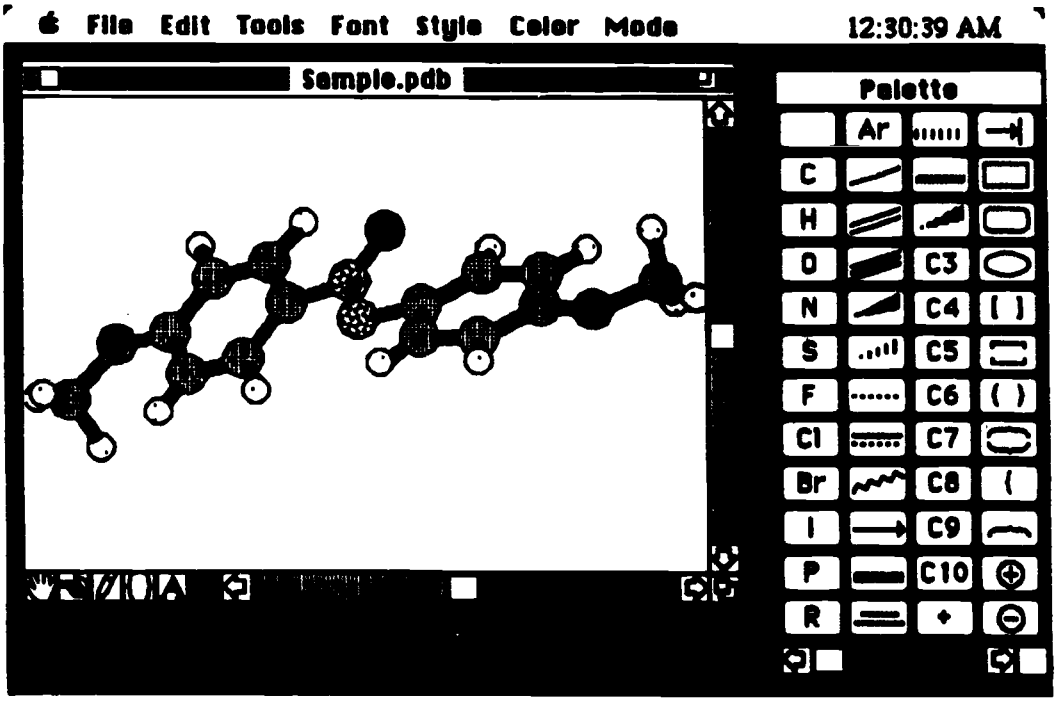


Figure 4. A ball-and-stick model of a protein. A portion of the menu/drawing palette is shown on the right.

Making the High School Varsity Team: Adding on to Basic Skills

One of the most important improvements in computer technology has been in the numerical processing capabilities of personal computers. Integrated circuit technology has improved to the point that it is possible to put powerful "chips" into small, (relatively) inexpensive desktop computers. Hence most computers come packaged with powerful mathematical engines that not only can perform complex numerical analyses but also generate or process data rapidly. The software technology, or the instructions to tell the hardware what to do, has not increased as rapidly, but useful and inexpensive packages are springing onto the market everyday. The problem we have, which is a good problem to have, is that so many new software products are coming onto the market that it is difficult to keep up with them.

Once the teacher and students are comfortable with the use of spreadsheets, limited-capability packages and menu-driven animation programs, the use of interactive simulation software and equation-solvers is a logical next step. Both of these platforms offer the student a "blank-slate" upon which to design a model or a simulation

of a chemical phenomenon. Both contain impressive mathematical engines with which to generate and visualize data. One of the nice features about this level of software is that the mathematics involved is often seen as a "black-box" to the student; in other words, the student doesn't know (unless she wants to) that the engine is doing a fourth-order Runge-Kutta integration of an ordinary differential equation or that the system is generating its answers using a rule-based algorithm based on artificial intelligence principles. Most of these programs allow the student to think about and concentrate on the variables of the system. Interactive simulation software is the class of products that allow a student to build a simple model of a chemical system, use the simulation to generate data, and then add more information or variables to the model.

Two different examples of interactive software are indicative of this category. Returning to organic chemistry, an example is the Beaker simulator published by Brooks/Cole Publishing Company. Beaker is a simulator that allows the student to solve general classes of problems. Students can draw organic molecules using the drawing tools, or by typing in the correct IUPAC name. The software can help the student to view the molecule in a number of different ways, including line segments, Kekule structures, and Lewis structures. The strength of the program, however, lies in the ability of the program to perform reactions based on the reactants that the student puts into the "beaker." The software, which is actually an artificial intelligence expert system, knows the "rules" for solving general classes of organic reactions. It will not be able to solve every possible combination of reactants, but it knows enough to be a solid teaching tool for young chemists. The visualization component, in addition to viewing structures, is that it will show all of the steps of the reaction mechanism, including multiple pathways if they exist. The student can change the reaction conditions, such as temperature, to see the effect on the reaction. Figure 5 shows a snapshot of a window in Beaker, in which an alkene is reacted with sulfuric acid and water to form an alcohol.

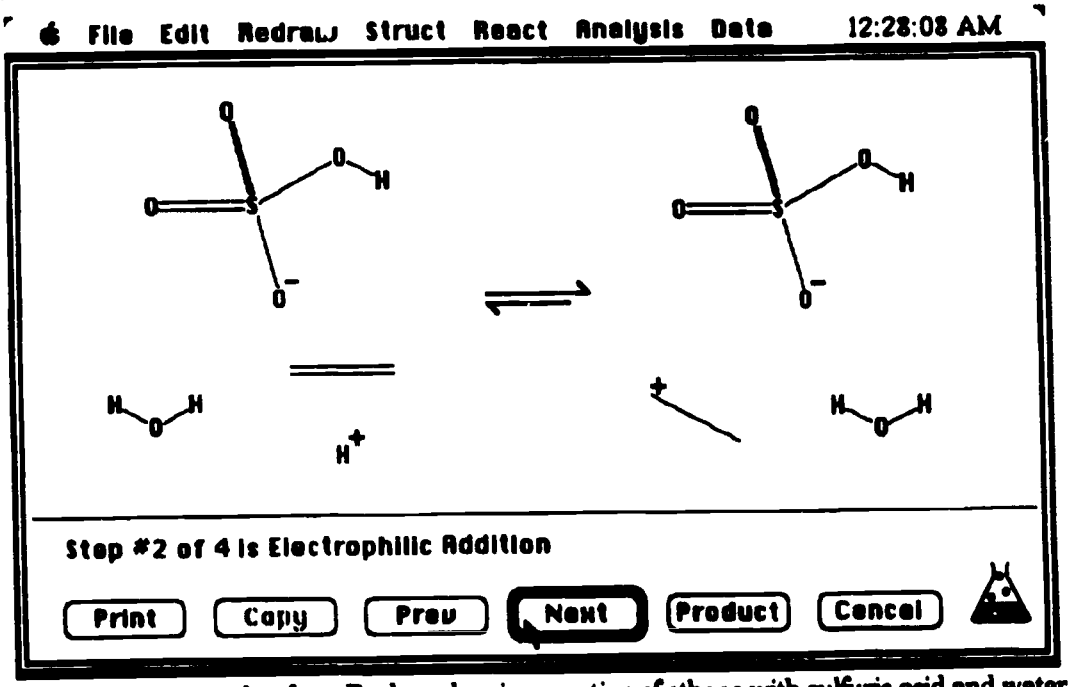


Figure 5. Screen shot from Beaker, showing reaction of ethene with sulfuric acid and water

Another type of interactive simulation software is STELLA, an icon-based simulation language. With STELLA, students can develop time-driven models by connecting different icons based on their relationship to each other. Figure 6 shows a sample model that simulates the decomposition of nitrous oxide into two components. By changing the rate constants, temperature or concentration of the reactant, the student can study the effects of those variables on the kinetics of the reaction.

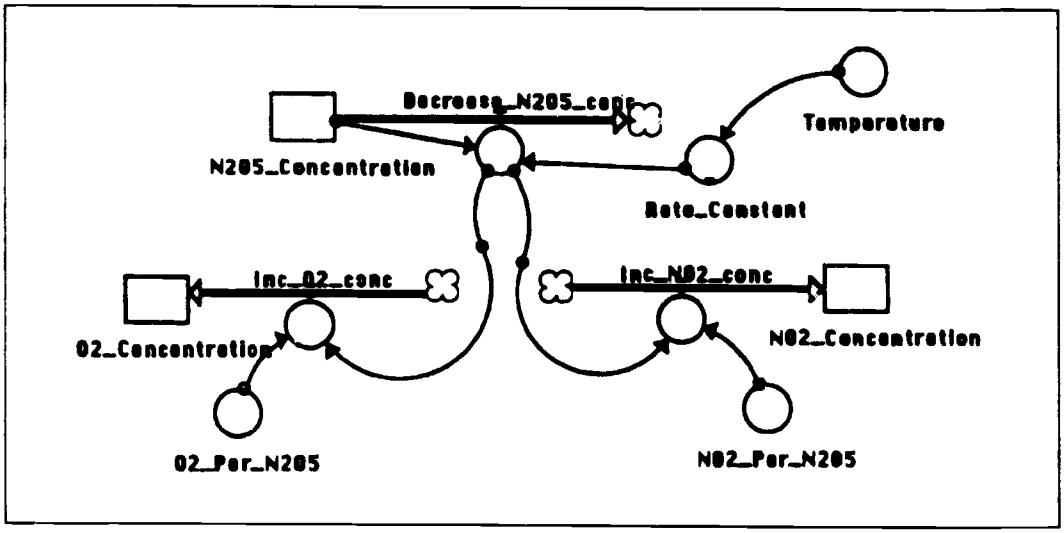


Figure 6. STELLA kinetics model for decomposition of nitrous oxide

One of the interesting and useful features of this software is that the software will graph several or all of the variables on a time graph while the simulation is running, so students can see changes while they occur. Figure 7 shows a graph generated for the kinetics model shown in Figure 6.

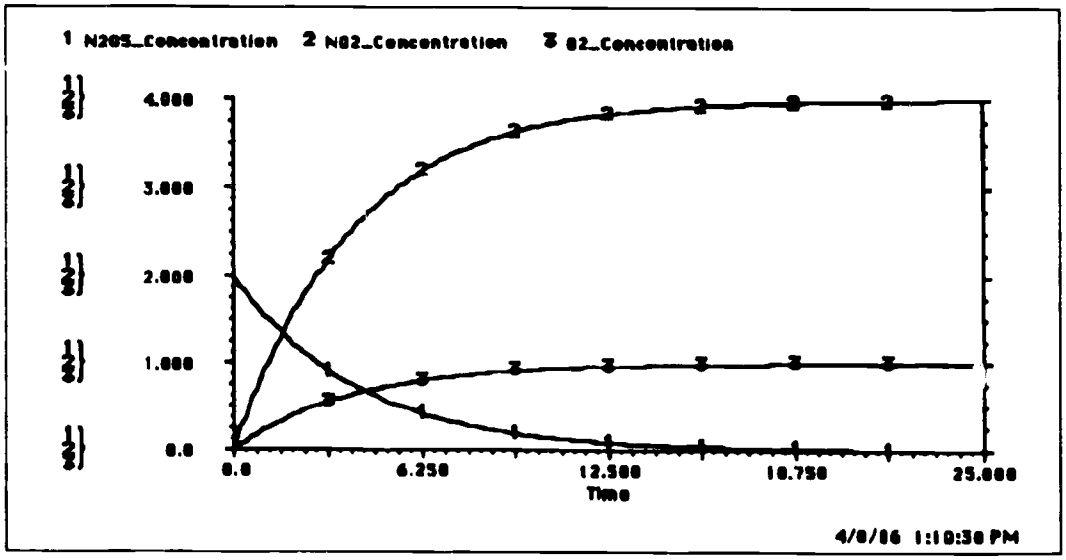


Figure 7. STELLA kinetics graph showing decomposition

The mathematical engine underlying this package is the use of Euler's method or Runge-Kutta techniques for solving ordinary differential equations. The mathematics are transparent to the student unless the student wants to study the mathematics. Students at Blair High School learn STELLA in the ninth grade, and are able to use the simulation software for a number of projects in physics, chemistry, and social studies. As STELLA also can generate a table of data which can be imported into a spreadsheet or other software package, faculty at Blair are looking at ways to let STELLA generate large amounts of data for visualization with more sophisticated graphics packages. STELLA, while intended to be used in educational institutions, is being used as a professional scientific research tool by a number of organizations, primarily in the environmental field. Students can develop simple models, then easily add on to make complex models as they begin to better understand the phenomenon being investigated. In our participation in the SuperQuest contest, we have used STELLA as a prototyping language to determine how well the student's model will work on a larger platform such as a supercomputer.

Equation-solvers are a new type of software for use in computational chemistry and computational science. Equation-solvers are programs which allow students to enter an algebraic equation such as PV=nRT, and then enter values for the known variables. The program will then perform the algebraic manipulation necessary to solve for the unknown variable. This differs from spreadsheets, in which the user must know prior to creating the spreadsheet which variable of the equation will be unknown. A simple equation-solver is TKSolver Plus, which comes equipped with graphing and table-generation capabilities. Figure 8 shows a sample screen from a TKSolver model of quantum mechanics, looking at the wavefunctions for a hydrogren atom.

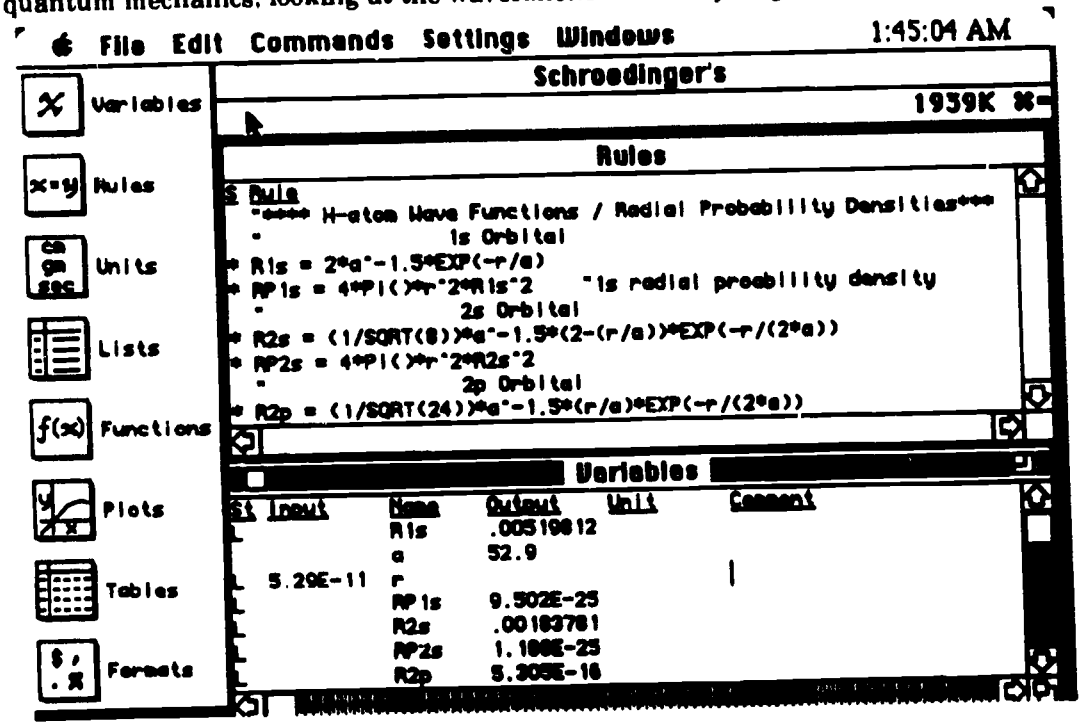


Figure 8. Screen shot of TKSolver Plus model of hydrogen wavefunctions

Figure 9 shows the graph generated for this model. TKSolver has an iterative solver as the mathematical engine; students can enter lists of data, and the software evaluates each data point until the calculations are completed. Output is written to an output list which can be graphed by the software or exported to another package. Figure 9 shows the graph of the data generated by the mathematical model.

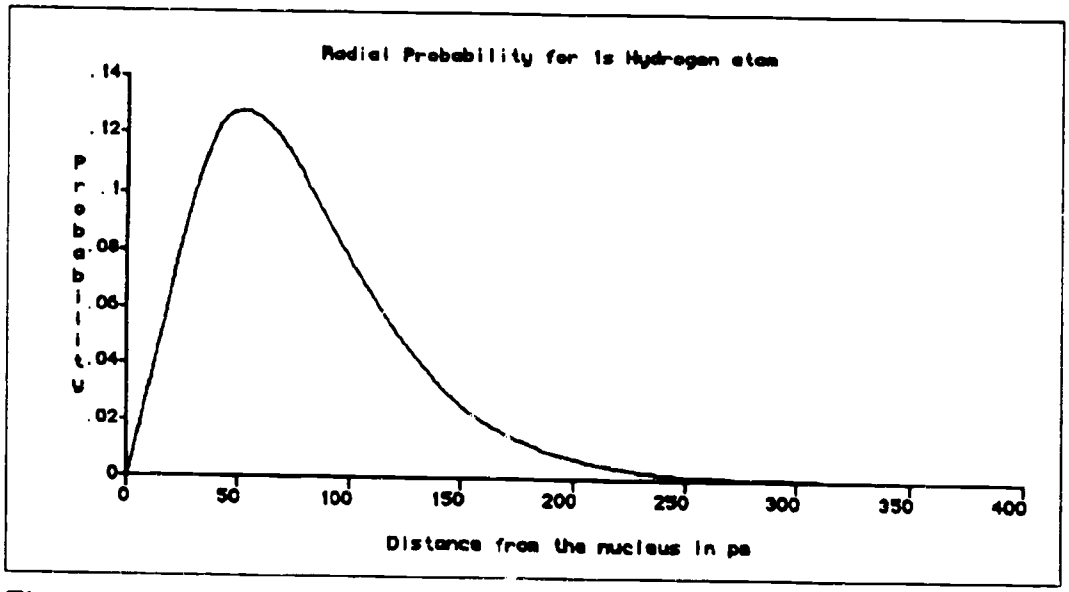


Figure 9. Graph from TKSolver Plus model, showing probability as a function of distance from the nucleus

TKSolver is a superb introductory equation-solver for students. The real excitement of equation-solvers, however, is in the form of packages such as Mathematica and Theorist. Both of these are very powerful mathematical tools for building mathematical models of physical events. In addition to their number-crunching abilities, both packages come bundled with two- and three- dimensional graphics capabilities and the abilitiy to create animations of events. Figure 10 shows a graphical representation of an electron orbital using Theorist, generated using the equation shown above the rendering.

A sample application in Mathematica would be the graphical creation of a box into which spheres representing molecules are placed. Figure 11 shows a sample frame from the animation. The user can manipulate the variables of the equation to show the three-dimensional behavior of the molecules as the temperature is raised or the pressure is reduced.

Blair faculty have used Mathematica for students who are in their first course in calculus; models are currently being built for use with advanced chemistry students in the area of atomic structure, kinetics, and thermodynamics. Mathematica was written with the intention of being a mathematical workbench for the practicing chemist, physicist, or mathematician. It is possible to use Mathematica without completely understanding the calculations being performed, as many calculations are established using built-in functions. Mathematica is an affordable package that runs on desktop personal computers. Several textbooks currently on the market provide good introduc-

tions to the use of this software and ideas and applications for teaching science and mathematics.

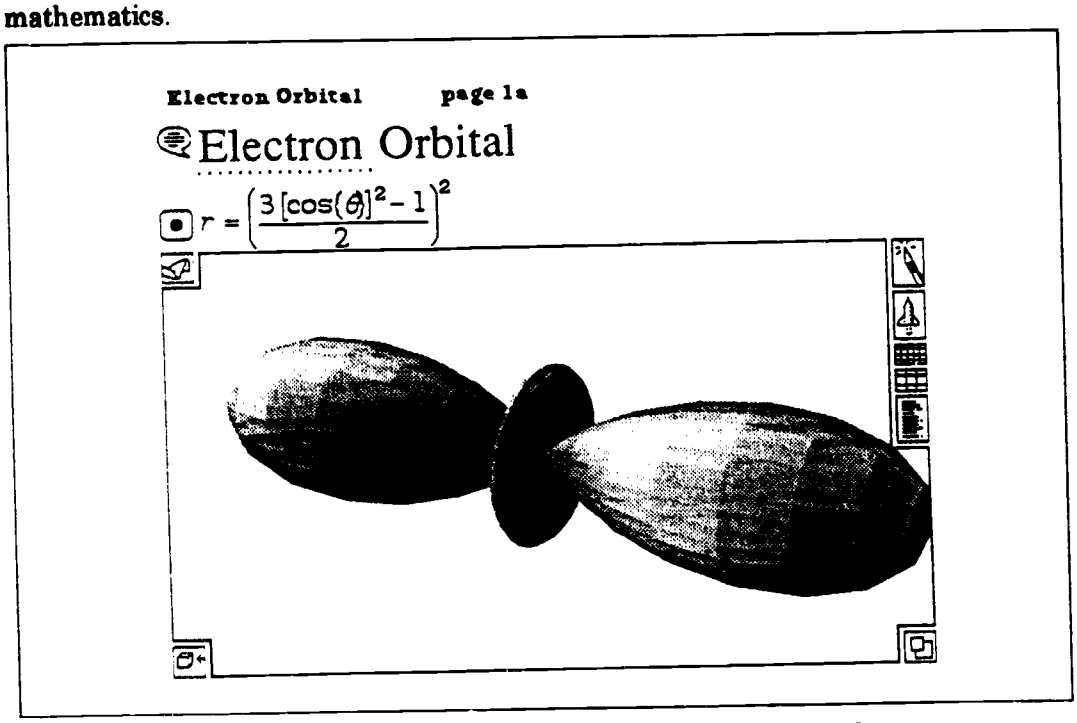


Figure 1C. Model of electron orbitals using an equation-solver to compute Schrodinger's equation

Graduating to College: Putting the basics to Work

A point that needs to be emphasized is that the tools described so far are very useful tools that should be able to support many projects that teachers or students might want to undertake. Many other packges, such as Hypercard for the Macintosh, also offer impressive possibilities for scientific visualization in the classroom. As mentioned in the introduction, there are many products not described here that could be useful for helping students to learn the concepts of computational science and scientific visualization.

There are two other types of visualization packages, however, that require more advanced skills for the user. One of these packages is that data visualization software which has been developed to run on personal computers but using large volumes of data generally collected from larger platforms. These packages have been designed to offer visualization capabilities to researchers who want to be able to look at their data while at their desktop computer. These packages, however, can be used successfully with secondary students.

Another type of package in chemistry is the molecular modeling software that has as its origin the programs found on supercomputers. These packages allow the student to perform sophisticated research or for the teacher to provide students with the

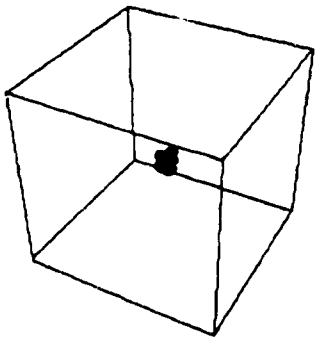
Ideal Gas

Ideal Gas

by Theodore W. Gray

This Notebook contains an animation of some particles bouncing about in a box. The example shows what would happen if, for example, a balloon containing some gas molecules was burst in a vacuum.

Example



To start the animation, select the rightmost bracket and press ⌘Y. This animation looks best run cyclically on a black-and-white screen.

This example was generated with the following code, which uses functions defined in the Implementation section below.

```
startingPoint = Table[{(Random[Real, {-1, 1}],
Random[Real, {-2, 2}]), (Random[Real, {-1, 1}],
Random[Real, {-2, 2}]), (Random[Real, {-1, 1}],
Random[Real, {-2, 2}])}, {20}];

walls = {{-12, 12}, {-12, 12}, {-12, 12}};

BouncePoint3[startingPoint, walls, 100]
```

Implementation

Figure 11. Frame from ideal gas simulation/animation using Mathematica

opportunity to delve deeply into the mysteries of atomic structure and reaction mechanisms. The most impressive package of data visualization software is the Scientific Visualization Software Suite developed by the National Center for Supercomputing Applications (NCSA). One of the best things about this package of software tools is that it is free; descriptions of the software and how to obtain it are found in Appendix B. This software is being used at the high school chemistry level by teachers working in conjunction with NCSA; one of the chapters of this monograph describes their work with simulating electron densities of lithium hydride molecules.

Components of the package such as NCSA Image can be used to display and animate scientific images. The user can easily change the colors of the image by selecting the desired palette. Animation is accomplished by developing frames, placing them in the same area of the disk, and selecting the animation option. Users can change the speed of the animation easily. Without a doubt, this collection of software tools is one of the most impressive on the market. The collection also included versions that can be run on larger platforms such as the Sun workstation or the X Window system for IBM platforms. The obvious advantage to this is that the student can learn the basics with relatively small data sets using the desktop version, then "graduate" to the more powerful versions when they are able to access a high-performance computer and graphics terminal. The IBM X Window system is the scientific workstation that was awarded by IBM to the 1992 winning SuperQuest team; other high schools throughout the country are getting time on these machines through cooperative efforts with local universitites or research institutions. The point is that it is becoming more commonplace for secondary science students to have the opportunity to use some of the more powerful scientific computing tools available to research scientists. Blair faculty are beginning to develop applications for the NCSA software suite with the desktop version and with the X Window version.

Molecular modeling software is abundant on the market; some smaller versions have already been described. Simpler packages will allow students to design molecules, perhaps rotate them or view them stereoscopically. There are now molecular modeling packages available for desktop computers that allow students to design and visualize molecules, and also to perform supercomputer-like calculations on the molecules. A classic example of a powerful modeling system is Chem3D, by Cambridge Scientific Computing. Chem3D will create single views, multiple views from different angles, and movies. The system is capable of handling large molecules, a feature not usually found with less-powerful versions. The key difference in this package is the ability of Chem3D to perform molecular mechanic calculations, such as computing bond stretching parameters, pi orbitals using self-consistent field theory (SCF), charge-dipole interaction terms, and other standard measurements. Chem3D can create connection tables or Cartesian coordinate files. A sample activity using Chem3D being used at the University of Maryland (O'Haver, 1992) is a lab which asks the student to answer this question using the graphics interface and some of the computational powers of the software: The boiling point of certain halogen-substitituted organic compounds varies according to the identity of the halogen. Does this variation correlate best with the electronegativity, the polarizability, or the sheer size of the halogen atom? This activitiy uses a variety of computational and graphics tools, including a three-dimensional version of the periodic table called "MacMendeelev", Chem3D, and Cricket graph.

Moving on to the Minor Leagues: Visualization on Large Platforms

One of the most important things to understand about visualization is the strength of the software available for personal desktop computers. Not only do many of them come complete with superb graphics capabilities or the ability to animate data, but the mathematical engines are often first-class. We have found, however, that students are highly motivated by the opportunity to work on larger platforms such as a supercomputer or a scientific workstation. The success of SuperQuest has been that students are using the same computational tools as research scientists. Students are also attracted to the

high-resolution graphics capabilities of the terminals connected to these machines. This section will profile some of the visualization tools that are available on larger platforms, such as multi-user workstations and supercomputers such as the CRAY and IBM ES/3090.

One of the most impressive examples of such a system is the CAChe (Computer-Aided Chemistry) package developed by CAChe Scientific, a subsidiary of Tektronix. This system has been included under this category of larger platforms because, even though it does run on higher model Macintosh computers, it does require special hardware in the form of a RISC coprocessor card, trackball, and special display terminal. Color Print 2 shows a stereoscopic view of the Vitamin B12 molecule. Using special glasses combined with a Tektronix monitor, the user is able to see a true 3-D view of the molecule.

This system can perform a wide variety of computational chemistry calculations on molecules, including MOPAC, Extended Huckel, ZINDO, and MM2. The software can patch to *ab initio* and semiempirical packages located on supercomputers, using the data generated by those platforms to look at the interactions of the molecules. CAChe can take standard file formats created by smaller systems such as Chem3D to generate data sets. The system allows the student/researcher to predict and visualize a number of chemical properties, such as electrophilicity, bond order, reaction pathways, activation energies, IR/UV/Vis spectra, solubility, heats of reaction, and stability. The user interface is Macintosh-like with pull-down menus. Color Print 3 shows a model of chromium, molybdenum, and tungsten hexacarbonyls with MM2 calculations on atom distances.

In addition to packages such as CACHe, which are complete molecular modeling packages, there are software packages which have the sole purpose of providing sophisticated visualization support. All of these packages run on larger platforms, such as scientific workstations or supercomputers. FieldView, produced by Intelligent Light, is an example of this type of software. FieldView is used to visualize large data sets of fluid dynamics problems, which are three-dimensional and time-dependent in nature. Color Print 4 shows a screen shot of a NASA AMES data set of the space shuttle, illustrating pressure, velocity and Mach number using contouring, vector field, iso-surface and cutting plane techniques. The platforms required to run such a package include the IBM RS 6000 series, SUN Workstations, and Silicon Graphics workstations. Students who are finalists in the SuperQuest competition are often presented with the opportunity and training to use some of these sophisticated graphics packages during the summer institute. High-quality graphics packages are usually included with the scientific workstations that each school wins if they become a finalist team.

The Major Leagues: Visualization on Supercomputers

Needless to say, there are very few high schools (one that I know of) that house a supercomputer in their facility. For many teachers, the possibility of being able to access and utilize a supercomputer is remote. Many supercomputer centers and universitities, however, are encouraged by the results of programs such as SuperQuest and are making their machines available to local schools. As a result, it is not unfeasible for a chemistry teacher to be able to use some of the capabilities that are possible only on these large computers.

Two types of software packages that require supercomputing power are briefly described in this section. The first is UniChem, a complete molecular modeling package found on the Cray supercomputer, and AVS, a state-of-the-art visualization package also developed for the Cray.212 UniChem is a package that allows the user to perform multiple molecular modeling and simulation techniques in a single integrated environment. UniChem is used for a variety of purposes, such as drug design, environmental studies, investigation of polymers, the study of agrochemicals, and investigations in material science. This package can incorporate many of the capabilities of most of the quantum chemistry codes, such as GAMESS, Gaussian, CADPAC, and MNDO90. With a built-in visualization system, this software allows the researcher to immediately show the generated data in a visual format.

During this school year, a full-day workshop was taught by a computational chemist from Cray to a group of high school teachers. After a morning of orientation on computational chemistry, the teachers were able to use the package during afternoon lab sessions to create and investigate a variety of chemical modecules. The teachers were able in a short period of time to investigate how geometries of molecules effect their reactions with other substances. UniChem, like many other packages, uses a Macintosh-like interface, so that the user does not have to know a programming language or be able to use a text editor to use the software. In a wrap-up session at the end of the workshop, the teachers were able to list several ideas for projects which might use the capabilities they had learned during the day. Virtually all agreed that the use of computational science and visualization was well-worth investigating as a potential teaching tool at the high school level.

There are a wide variety of visualization software packages available on supercomputers. Several examples are Stardent's Application Visualization System (AVS) and a visualization package by Wavefront. Both of these products have fairly high learning curves, and are usually operated by professional visualization technologists. The general scenario for the use of these products is that a researcher will generate data using a supercomputer, then meet with the visualizer to discuss what the researcher wants to see. The visualizer then begins the complex task of applying the software to the data set. In a collaborative manner, the scientist and visualizer continue the editing process until the researcher has a finished visualization product. Many times the results are produced in a videotape format, which the researcher can use at professional meetings. While this type of visualization may seem to be unaccessible to secondary students, we have had a number of students from past SuperQuest competitions utilize this service. Once they had generated some data, they met with the Wavefront technicians at Cornell to produce short-segment videotapes. The students were able to use these videotapes as a part of their final presentations at SuperQuest, as well as in other scientific contests such as Westinghouse and the International Science Fair.

As stated above, many of the supercomputer centers that we have worked with at Blair, such as Cornell and NCSA, have been very willing to work with students on computational projects. Another supercomputer center in North Carolina, which sponsors its own state-wide SuperQuest, has recently been awarded a contract to become the international visualization center using the AVS software. This will be a wonderful resource for secondary students in that state.

Visualizing the Future

Incorporating computational science and visualization into the secondary science class requires resources and, more importantly, time. Both are things which are in short supply in the lives of most high school teachers. The use of computational science also requires a change in mindset about how we teach chemistry and what is important for future chemists.

I became convinced of the need to re-think my teaching through SuperQuest, and the many contacts I made with practicing chemists through the SuperQuest experience. I was also convinced with the initial success I had in using some of the tools described above with my students at Blair, beginning with the ninth-graders. We've been fortunate with our success with SuperQuest; our victories have brought state-of-the-art hardware into our building. But the majority of the work we have done to date have utilized our desktop computers and free or inexpensive software. The use of simulation software with its built-in visualization functions has made it much easier to help my students see the dynamic nature of chemical systems. I feel much less constrained by the limitations of the lab, in terms of time, safety, and expense. While not neglecting the lab, we've been able to have the students look at systems that would not be possible in the lab. We still (occasionally) use videos of chemists doing demonstrations, but they are not as effective as simulations that the students can manipulate themselves.

Our use of computational science at Blair was experimental at first, but it is now a solid component of almost all of our science and math courses. We continue to develop new activities and modules. We are hoping that as more schools become involved in computational science and visualization that they will develop activities and share them with others. Cornell University is making strong efforts to help secondary teachers become involved in this area (using many of the materials developed at Blair); the North Carolina Supercomputer Center and NCSA are similarly involved. Contacting these centers is certainly a good way to begin your efforts in learning about computational science and visualization. In conclusion, a statement by Wolff (1991, p.236) summarizes where science is going as we head into the next century:

> My own belief is that scientific visualization is the starting point for a major paradigm shift in the way that people in all areas deal with large amounts of information. Multimedia technologies form a natural bridge between the compute- and data-intensive world of the researcher and the video-oriented presentation technologies that are commonplace in classrooms and scientific meetings. In the not-too-distant future, researchers and teachers will deal with images and animations as easily as they now handle text-based documents in word processors.

References

Boyd, Donald (1992). *Reviews in computational chemistry*. New York, NY: VCH Publishers.

Hirschy, L. (1990). Computer package puts chemistry at your fingertips. *Research and Development*. Cahners Publishing Company.

National Science Foundation (1987). Visualization in scientific computing. *Computer Graphics, 21*(6).

O'Haver, T. (1992). Personal communication, March 10, 1992.

Steinhorn, L. (1992). Whiz Kids in America. Washington, DC: *Washington Post.*
Wolff, R. (1991, May/June). Visualization as a tool for physics education. *Computers in Physics,* pp. 278-286.

Appendix

Software for Computational Chemistry and Visualization

Most of these listings come from Boyd (1992). Many of these vendors will offer helpful ideas on getting started. It is also valuable to consult the *Journal of Chemical Education* computer section.

PERSONAL COMPUTERS: Apple Macintosh, IBM PC/PS2, IBM PC/AT/XT

STELLA
High Performance Systems
13 Dartmouth College Highway
Lyme, NH 03768

Simulation language for the Macintosh, based on manipulation of icons. Mathematical engine solves differential equations based on a time-driven simulation

TKSOLVER Plus is available through the American Chemical Society

Mathematica and Theorist are commercial products available through most retail software outlets.

MM2, CNINDO/D, FORTICON8, MNDO, HAM/3, POLYATOM, MOPAC, DRAW, MOLVIEW, NAMOD, etc.

Quantum Chemistry Program Exchange (QCPE)
Department of Chemistry
Indiana University
Bloomington, IN 47405
812-855-4784

Extensive catalog of programs for quantum mechanics, molecular mechanics, and molecular graphics. QCPE being academically affiliated provides software and documentation at nominal cost. PC and Mac II.

PCMODEL
Serena Software
Dr. Kevin E. Gilbert
P.O. Box 3076

Bloominton, IN 47402
812-333-0823

Structure building, manipulation, energy minimization by MMX, stick and dot surface display, derived from MODEL, handles inorganic as well as organic molecules, also models transition states. Companion MOPAC 4.0 program. Mac II. IBM PC/XT/AT, Silicon Graphics IRIS, Apollo versions.

CHEM3D PLUS
Cambridge Scientific Computing, Inc.
Dr. Stuart Rubenstein
875 Massachusetts Avenue, Suite 41
Cambridge, MA 02139
617-491-6862

Structure building, manipulation, simple force field and MM2 energy minimization, ball-and-stick and space-filling display. Mac II.

NITRO
Tripos Associates
1699 Hanley Road
St. Louis, MO 63144
800-323-2960

Graphics processor for interfacing to SYBL on a VAX. Mac ii, PC versions. Alchemy does structure building, manipulation, SYBYL energy minimization, stick or space-filling display on a PC. Alchemy III interfaces to Chemical Abstracts Service registry files.

CHEMCAD +
C_Graph Software, Inc.
P.O. Box 5641
Austin, TX 78763
512-459-3562

Structure building, manipulation, van der Waals and electrostatic energy minimization by MM2 and MNDO, stick or ball-and-stick display, report generation, interface to ChemFill. PC. Also Macintosh versions of MM2+ and MNDO+.

MOLIDEA
CompuDrug USA, Inc.
P.O. Box 202078
Austin, TX 78720
800-877-0880

Structure building, manipulation, van der Waals and electrostatic energy minimization, CNDO/2 and other simple MO calculations, interfaces to packages for log P and statistics, stick, space-filling, or dot surface display. QSAR and expert systems program. PC.

CAMSEQ/M
Weintraub Software Associates, Inc.
P.O. Box 42577
Cincinnati, OH 45242

Structure building, manipulation, rigid conformational searching with interface to CAMSEQ/PC, stick, ball-and-stick, and space-filling display. PC.

MICROCHEM
Chemlab, Inc.
1780 Wilson Drive
Lake Forest, IL 60045
312-996-4816

Structure building, manipulation, energy minimization of organic, inorganic, and polymer units, stick, ball-and-stick, and space-filling display, QSAR Craig plots. Mac.

DESKTOP MOLECULAR MODELLER
Oxford Electronic Publishing
Oxford University Press
Walton Street
Oxford, OX2 6DP
U.K.
44-865-56767 x4278

Structure building, manipulation, energy minimization, stick, ball-and-stick, space-filling display. PC.

CACHe
CACHe Group
Tektronix, Inc.
P.O. Box 500, Mail Stop 13-400
Beaverton, OR 97077
503-627-3737

Structure building, manipulation, MM2 energy minimization, stick, ball-and-stick, or space-filling display, extended Huckel molecular orbital, electron densitities, and electrostatic maps, 3D viewing. Tektronix enhanced Mac II.

Minicomputers, Supercomputers, Workstations

MM2, FORTICON, CNINDO, CNDO/S, PCILO3, MOPAC, AMPAC, MNDOC, GAUSSIAN, HONDO, DISGEO, ECEPP2, etc.

Quantum Chemistry Program Exchange (QCPE)
Department of Chemistry
Indiana University
Bloomington, IN 47405
812-855-4784

Extensive catalog of programs for quantum mechanics, molecular mechanics, and molecular graphics. QCPE being academically affiliated provides software and documentation at nominal cost. All large workstations, minicomputers.

MacroModel 2.5, 3.0
Dr. W. Clark Still
Department of Chemistry
Columbia University
New York, NY 10027
212-280-2577

A user-friendly molecular modeling package for molecular mechanics and conformational searching of organic molecules, proteins, nucleic acids, carbohydrates, AMBER- and MM2-like force fields, implicit solvation model, reads Cambridge and Brookhaven PDB files. VAX, Convex, Alliant, and workstations.

SYBYL
Tripos Associates
1699 Hanley Road
St. Louis, MO 63144
800-323-2960

A complete, user-friendly molecular modeling package with capabilities for molecular mechanics, conformation serching, minimization, semiempirical and ab initio MO calculations, molecular graphics, active analog approach. Tripos, AMBER- and MM2-like force fields. Components for handling organic molecules, macromolecules, and polymers. Interface to Cambridge Structural Database and Brookhaven Protein Database. CONCORD rule-based molecular model builder. QSAR module based on comparative molecular field analysis.

CHEM-X
Chemical Design Inc.
200 Route 17 South, Suite 120
Mahwah, NJ 07430
201-529-3323

An integrated, comprehensive set of modules including ones for building of organic and inorganic structures, graphics, conformational analysis, quantum mechanics, database management, statistics (QSAR), proteing modeling, dynamics.

QUANTA/CHARM
Polygen Corporation
200 Fifth Avenue
Waltham, MA 02254
617-890-2888

Structure building, manipulation, energy minimization, and molecular dynamics, Boltzmann jump Monte Carlo conformational searching, protein homology search ing, MOPAC interface. QUANTA is an interactive graphics front-end to empirical energy calculations using the Chemistry at Harvard Macromolecular Mechanics force field. Reads Cambridge and Brookhaven PDB files. Polymer package.

BIOGRAF
BioDesign, Inc.
199 S. Los Robles Avenue, Suite 270
Pasadena, CA 91101
818-793-3600

Structure building, manipulation, energy minimization and molecular dynamics in VAX or workstation environment. POLYGRAF for treating polymers.

INSIGHT/DISCOVER
BIOSYM Technologies, Inc.
10065 Barnes Canyon Road, Suite A
San Diego, CA 92121
619-458-9990

Structure building, manipulation, energy minimization and molecular dynamics, protein loop searching, MOPAC interface. INSIGHT is an interactive graphics front-end to the empirical energy calculations of DISCOVER. DMol for quantum mechanical density functional theory calculations. DelPhi for electrostatics potential maps.

CHEMLAB-II
Molecular Design, Inc.
2132 Farallon Drive
San Leandro, CA 94577
415-1313

A modular molecular modeling package for conformational analysis, quantum chemistry calculations.

PROPHET

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02238
617-873-3353

Molecular mechanics and display, statistical and mathematical modeling.

HYPERCHEM

Hypercube, Inc.
16 Blenheim Road
Cambridge, Ontario N1S 1E6
Canada
519-622-0260

Model building and display on a DOS-compatible Chemputer parallel processor; interfaces to molecular mechanics, semiempirical, and ab initio packages.

AMBER 3.0

Dr. Peter A. Kollman
Department of Pharmaceutical Chemistry
University of California
San Francisco, CA 94143
415-476-4637

Assisted model building using Energy Refinement. Energy minimization, molecular dynamics, and free energy perturbation calculations.

GAUSSIAN

Gaussian, Inc.
Dr. David J. Moses
Dr. John A. Pople
4415 Fifth Avenue
Pittsburgh, PA 15213
412-621-2050

GAUSSIAN86 and GAUSSIAN88 ab initio quantum mechanical calculations, archival storage of computed results, wave functions, configuration interaction geometry optimization, properties.

GAMESS

Dr. Michael Schmidt
Dr. Mark Gordon
Department of Chemistry
North Dakota State University
1301 12th Avenue North
Fargo, ND 58105
701-237-8906

General Atomic and Molecular Electronic Structure System; public domain software for ab initio calculations. Wavefunctions, properties, geometry optimization.

GRADSCF

Polyatomics Research Institute
Dr. Andrew Kormornicki
1101 San Antonio Road.
Suite 420
Mountain View, CA 94043
415-964-4013

Ab initio calculations, geometry optimizations, derivative properties, such as force constants and vibrational spectra.

HONDO

IBM Corporation
Dr. Michael Dupuis
Scientific and Engineering Computations Department
Department 48B, Mail Stop 428
Kingston, NY 12401
914-385-4965

Ab initio calculations for IBM 3090 and other IBM computers, interface to molecular graphics package for IBM workstations.

CADPAC

Dr. Roger Amos
Dr. N.C. Handy
Lynxvale WCIU Programs
20 Trumpington St.
Cambridge CB2 1QA
U.K.
44-223-336384

Cambridge Analytical Derivatives Package.

Chapter 9

The National Education Supercomputer Program

RICHARD ENDERTON
BRIAN LINDOW

What would happen to the earth's precipitation if the ozone level were radically altered, or if the earth's orbit were more elliptical, or if other land masses were formed? What would it look like if the function y=sin2t were graphed over time? What would happen if the index of refraction for a magnifying glass were changed? What happens when a proton moves through a magnetic field, and what would happen if its direction of movement were changed? What if...

Science and math teachers are intimately familiar with the myriad of "what if" questions that arise in the course of teaching their subjects. They are also familiar with the frustration of being limited to theoretical responses, when what they would most like to be able to do is demonstrate the answers, or better yet, have the students themselves try to discover the answers.

In fact, the "what ifs" in laboratories around the world are being approached quite differently by scientists today than in the past. Science has traditionally been a shuttle run between theory and experimentation, as shown in Figure 1. As illustrated in Figure 2, however, with the increase of speed and memory of computers, science has sprouted a third leg: simulation.

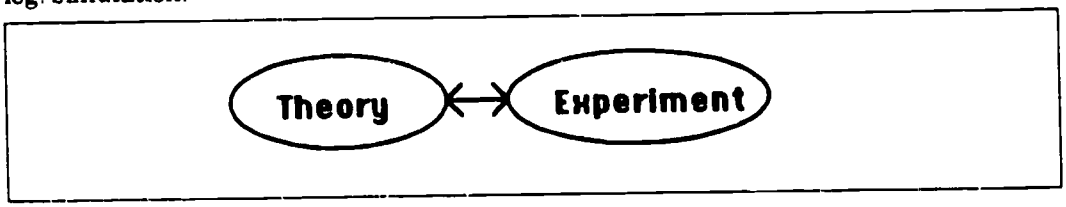


Figure 1. Science in the past

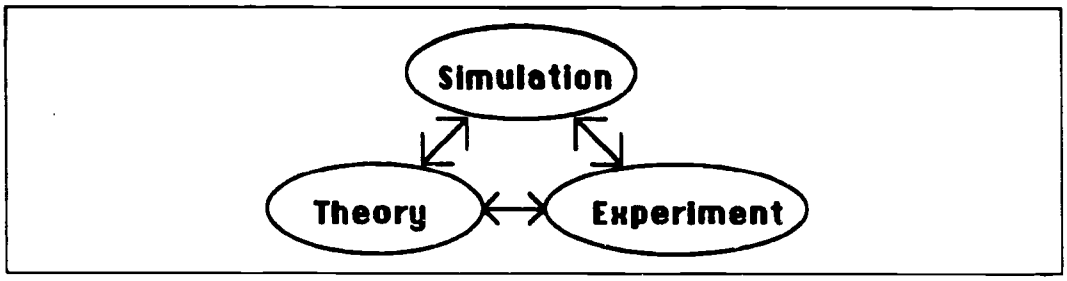


Figure 2. Science in the present

Many natural phenomena can be described by (sometimes extremely complicated) mathematical formulas and relationships. High speed computing has enabled scientists to examine such relationships and their results, through programmed simulations that the scientist can provide with a set of initial parameters. Experiments which would be difficult, if not impossible, to perform in real life can be simulated by a computer which does multiple repetitions of the formulas dictating how the elements of the experiment will interact. The results are then displayed, often using intricate graphic representations. Computer simulation has had a huge impact on the way science is performed at top-level laboratories, and now secondary schools have a chance to utilize some of these same simulation tools.

The National Education Supercomputer Program (NESP) is placing the power of supercomputers and their ability to do modeling and simulation in the hands of high school teachers and students around the country. NESP makes classroom demonstrations and experiments possible in ways that were, until recently, limited to large scale laboratories and top-flight scientists.

The National Education Supercomputer Program

NESP is sponsored by the U.S. Department of Energy, and has a mission to help America's teachers motivate and educate the nation's students in math and science. The program provides a unique instructional tool for science and math teachers and students in junior high school, high school, and community college classrooms throughout the U.S. Toward that end, the program goals are to:

- Perform large-scale computer simulations and modeling to enhance classroom science and math instruction and to demonstrate the importance of supercomputers in scientific research;
- Supply a reliable, progressive computing environment that supports efficient, wide-range network connectivity;
- Develop distributed software that allows students and teachers nationwide to use their local microcomputers in conjunction with the National Education Supercomputer;
- Explore and develop projects that foster interdisciplinary education;
- Provide a mechanism to facilitate communication between teachers nationwide and to encourage sharing curricula and ideas among teachers, students, and scientists.

The National Education Supercomputer (NES) is a Cray Y-MP, which was donated by Cray Research, Inc. It is housed at the National Energy Research Supercomputer Center (NERSC) at Lawrence Livermore National Laboratory (LLNL). The Department of Energy

provides funding for the program. Brian Lindow, L-561, Lawrence Livermore National Laboratory, P.O. Box 5509, Livermore, CA 94550 coordinates the program, and is the contact person for information concerning all aspects of NESP.

NESP consists of two basic components:

- The High School Teachers Supercomputer Workshops, and
- The High School Science Students Honors Program in Supercomputing.

Supercomputing Workshops for Teachers

High school teachers are given hands-on training with the NES in two-week workshops. During the course of the workshops, the teachers have the opportunity to learn the techniques of distributed computing and to become familiar with the applications and utilities afforded on the NES. Time is also spent developing curriculum applications of the tools available. Teachers have the opportunity to bridge the gap between textbook concepts and actual applications by working with these advanced scientific tools.

After training, teachers return to their own schools where they establish accounts for their students on the NES. The students can then also participate in simulations and modeling similar to the "big science" conducted at major laboratories. Access to the NES is accomplished via phone lines and modems from the classrooms, and electronic links to other schools participating in the program are established through the NES.

An outstanding aspect of the program is that it is not designed for just a few unusually talented students. The applications can be used without any knowledge of programming, and the steps involved in transferring files to and from the NES are elementary. Average students can become proficient in the use of the applications in a very short time. Then they can concentrate on the science which they are exploring. The applications, however, are only the beginning. There is literally no end to how far students and teachers can pursue their interests in fields which would otherwise be restricted because of a lack of computing power.

Best of all, the cost is minimal. Most schools already have the necessary hardware: microcomputers (IBMs or MacIntoshes), modems, and phone lines. Access to NESP is via an 800 number. Both the computing time and the application software on the Cray are supplied by NES. There is only the negligible cost of the file transfer software (which can be obtained as shareware).

The High School Science Student Honors Program in Supercomputing

The NESP has for the last eight years conducted an annual "Superkids" program which brings some of the nation's most talented high school math and science students to LLNL. Participants in this program, one from each state, are chosen by the state's governor. Students from several foreign countries have participated as well. The Superkids spend two weeks at Livermore with all expenses paid. LLNL employees serve as host families. During this period they tour many of the research facilities at the lab, as well spend time talking with the scientists and engineers at the lab. They also have opportunity to meet and hear some of the the nation's most eminent scientists.

The Superkids spend a portion of the two weeks learning supercomputer applications and developing projects to present on the final day. NERSC staff serve as lab advisors, and the Superkids gain a keen insight into the workings of a scientific community.

Information and names of contact persons for the Superkids Program in each state can be obtained from Brian Lindow.

NES and Distributed Computing

The underlying concept behind using the NES in the classroom is distributed computing. Each type of computer in the distributed computing environment is utilized according to what it does best. Microcomputers, virtually the only types of computers available in high schools, are excellent tools for seeing immediate results on the screen in response to information entered. Parameter entry for NESP applications is therefore done locally on microcomputers using interactive software. Much of the information is entered using a mouse, although numerical values are easily entered as well. As data is entered, the effect that the new information will have is displayed graphically on the microcomputer video screen. When the parameters are set, and the user has the desired setup, then it is time to tap into the NESP Cray's computing power.

The file containing the setup information is transferred to the NES via a modem and phone lines. Once the Cray receives the information, the appropriate application is run, with the setup file as input. The Cray supercomputer performs the enormously complex calculations, and produces an output file, which normally is a graphic representation of the results. Supercomputers are not, however, well suited to displaying graphics, so the output is returned from the Cray to the microcomputer via a modem and phone lines for viewing, once again utilizing the smaller computer for a job it is well designed for.

The distributed computing cycle of local set-up, calculations on the NES Cray, and local viewing, shown in Figure 3, is a very efficient use of resources. Many students can be working on experiment design and set-up, as well as interpreting results on local computers without tying up the NES. Even with only one phone line, students can take turns sending files to the Cray for computation while others are either performing setups or analyzing computed results, since file transfer time and computation time is relatively short.

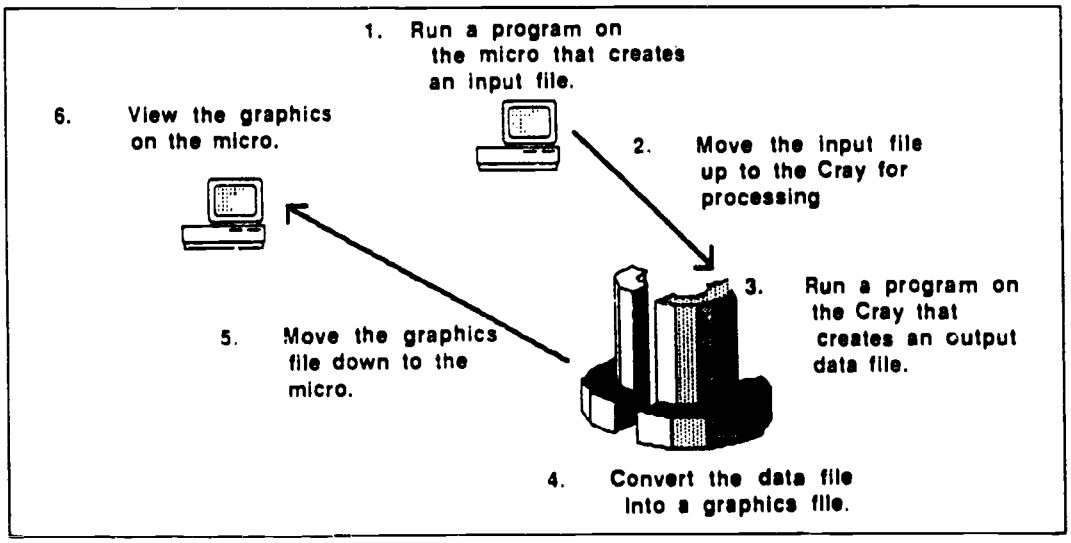


Figure 3. Distributed software cycle

BEST COPY AVAILABLE

Applications

At present there are three applications on the NES for which the accompanying setup and viewing software are available for microcomputers: ray-tracing, particle physics and climate modeling.

Ray Tracing (Wireman / Movie)

Ray tracing is an advanced technique for creating realistic graphic images of solid geometric objects in order to get a better appreciation for how they look. The benefit of computer graphics techniques is that solid models can be viewed and manipulated without the need for constructing an actual physical mock-up. Ray tracing techniques are used in a variety of commercials and movies as well as scientific modeling. Ray tracing builds an image by approximating the way light interacts with matter, calculating what happens to light rays as they strike objects which have colors, reflective and transparent qualities, surface texture and other attributes (see Figure 4).

The ray tracing program used on the NES is called Mesa. Mesa uses the principle of backward ray tracing (see Figure 5) to improve efficiency. Instead of calculating the path of light rays coming from the light source and interacting with objects, the calculations are done "backwards" from the eye position, which reduces the number of necessary calculations.

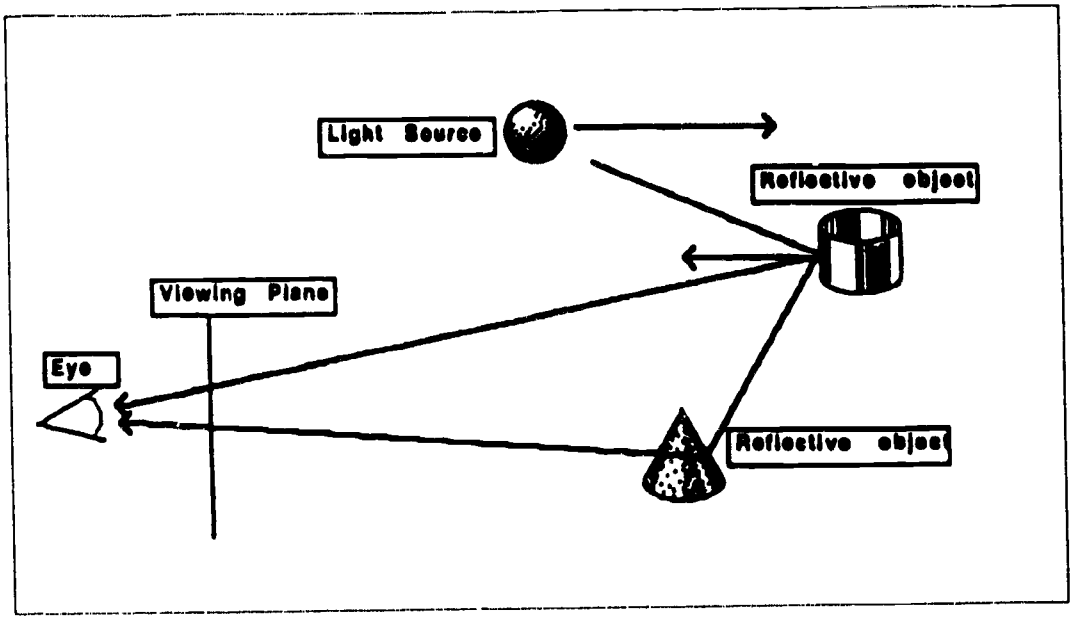


Figure 4. Forward ray-tracing

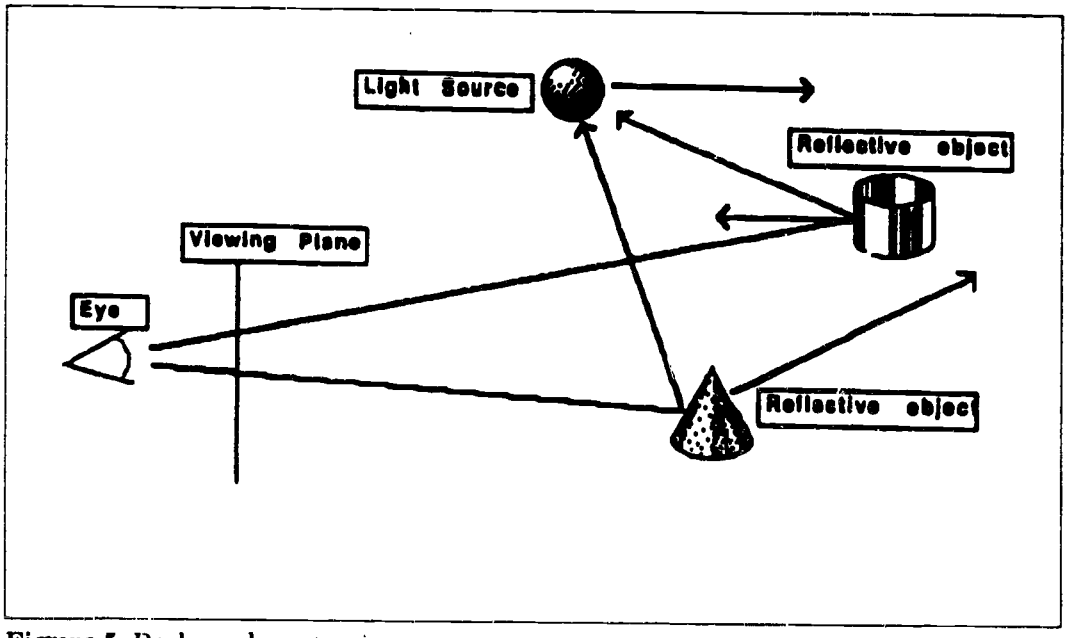


Figure 5. Backward ray-tracing

There are seven basic objects used in Mesa: sphere, cone, torus, cylinder, super spheroid (rectangular solid), square, and height field, as shown in Figure 6. Each of these objects can be given surface attributes, which include color, degree of opaqueness, transparency (with variable index of refraction) and reflectivity. The objects are placed in a three dimensional coordinate system, and they can be scaled and rotated and positioned using values oriented on the x, y and z axes.

Other parameters can also be altered. Eye position and focal point can be moved. Light sources can be added or moved, and they can also be colored. There are special attributes for some individual objects; for example the inner radius of the torus can be changed relative to the outer radius.

A height field is a square divided into a 512 by 512 grid. Values can be established for the elevation of each gridpoint. These values can be entered into a corresponding 512 by 512 array, usually according to a formula.

Images can be mapped onto the surface of objects. Patterns or pictures can be "pasted" to squares, spheres, or any of the other objects.

"Wireman" is the name of the program used on the microcomputer to enter the objects and their attributes. As shown in Figures 7 and 8, the screen consists of windows in which objects can be added, positioned, rotated and scaled. Eye position and viewpoint (focus) can also be established using a mouse. Pull-down menus contain options for altering the attributes of the objects and the light sources.

In the center of the screen is a viewing window in which a wireframe representation of the selected objects is displayed. As the objects are manipulated, the relative positions and scaling are updated on the viewing window. The wireframe objects display no color, but they do enable the user to constantly see a sketch of the scene as it is built and where the objects will be in the final result.

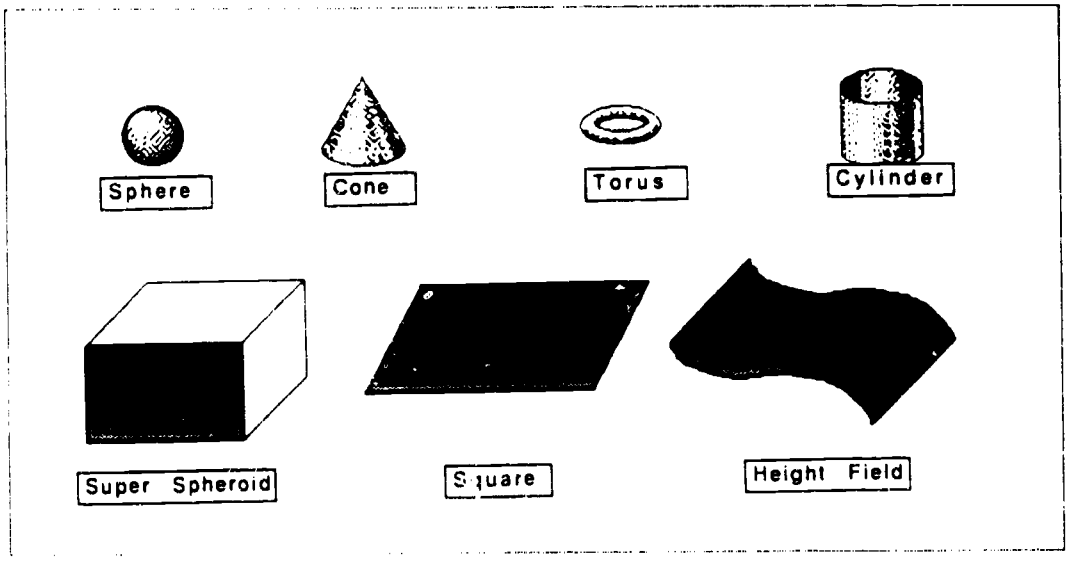


Figure 6. Objects in Mesa

After the scene is created, the user takes a "snapshot" of the objects. The snapshot records the objects' positions and attributes in what is called a snapshot file. Animations are easily created, in that a series of snapshots can be taken as object and/or eye position are altered. These subsequent snapshots are appended to the snapshot file. The snapshot file is then sent to the Cray supercomputer for processing. In the case of graphics generation,

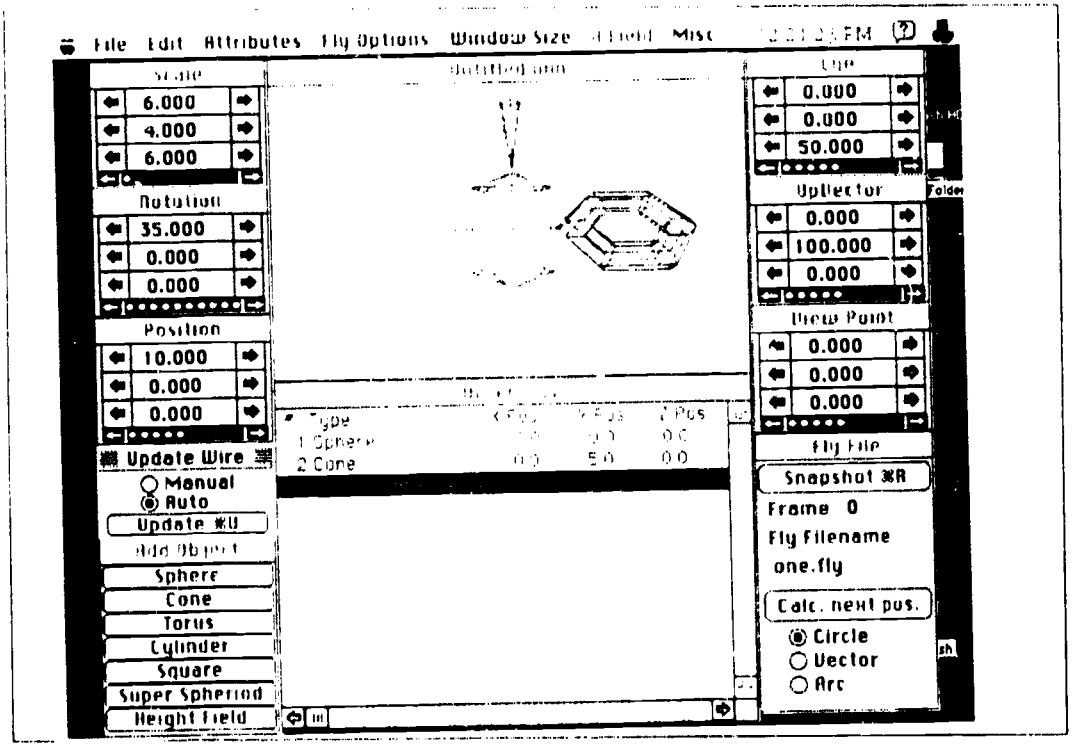


Figure 7. Wireman Interface

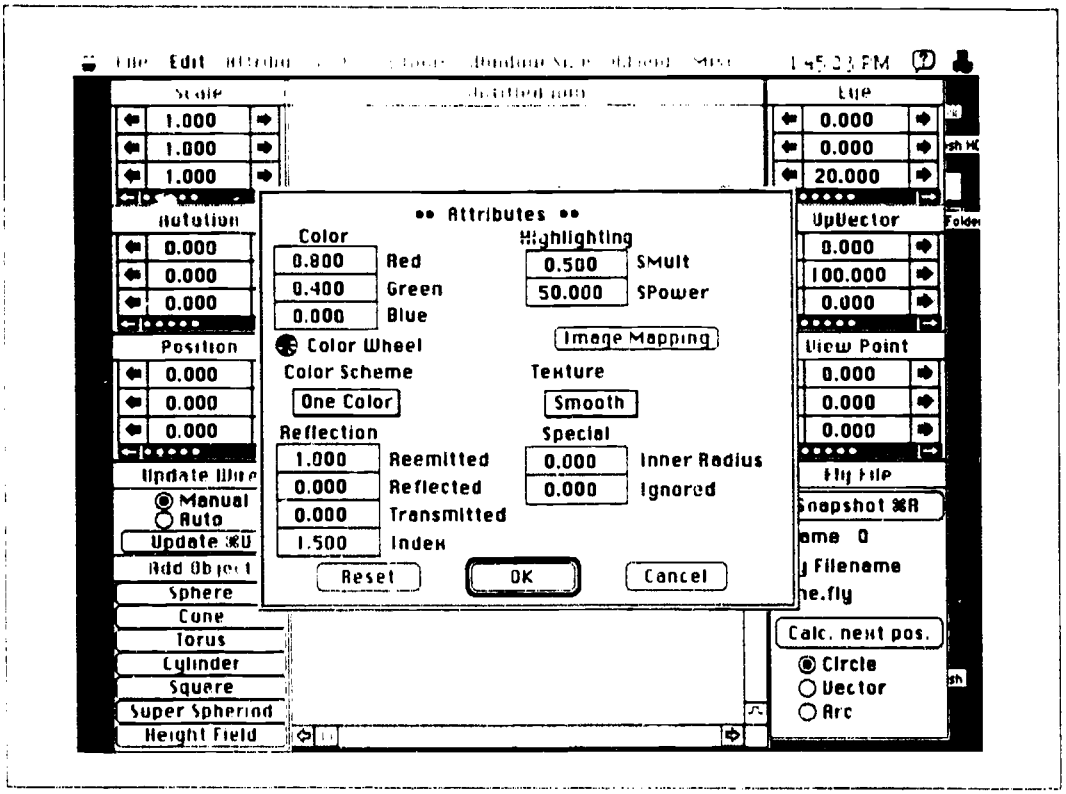


Figure 8. Wireman Attributes Menu

this processing is called "rendering." Rendering turns the snapshots into final, finished pictures. The wireframe sketches are filled in to make solid objects that have the color, shading, and other attributes derived from the application of the light sources in the model. The result of rendering the snapshot file is a graphics file, which is then returned to the microcomputer for viewing.

Ray tracing is a fascinating process, and students are intrigued by it. There are several inherent mathematical concepts in ray tracing of which the students gain understanding simply by immersion in the application. For example, when using Wireman, students work in a world determined by the three dimensional coordinate system. To manipulate objects, they must move around in the system, do scaling on the objects, and apply rotations, as well as keep track of where objects are relative to each other. They can use ray tracing to move further into mathematics by, for example, graphing functions over three dimensions, modeling slicing as applied to solids in calculus, and experimenting with fractals. Scientific applications can include such exercises as experimentation with lenses and the index of refraction and the modeling of scientific concepts such as molecules, atoms, and cells. Figures 9, 10, and 11, and Color Print 5 provide illustrations of the kind of visualizations that are possible.

As noted earlier, after the snapshot file is rendered on the Cray, the rendered output file is sent back to the microcomputer. The returned data is in the form of an 8-bit graphics file. Another program on the microcomputer, called Movie, is used to display the graphics file on the screen. If a series of snapshots were rendered, the first of these appears on the

screen. A VCR-type set of buttons is also displayed, so that the snapshots can be run in order forward or backwards, looping around from the last frame to the first and running non-stop. The frames can also be changed one at a time. Movies can be scripted so that text can be included on the screen. Scripts enable the user to show chosen frames, slow down the action, or chain several movies together into one longer movie.

Particle Physics Simulations (Particleman / PC3DV)

The particle physics simulator is based on a very simple model. The user creates a box, and into this box places up to hundreds of particles. Then the particles are released and the user can essentially sit back and watch what happens.

The box can be given whatever size is appropriate for the simulation. It can be big enough to encompass a solar system, or small enough to view interactions on an atomic level. Exit conditions for particles reaching the limits of the box can be set to loss (the particle simply disappears), periodic (the particle enters the box again on the opposite side), or bounce (self explanatory). Conditions in the box can be altered by changing the gravitation-

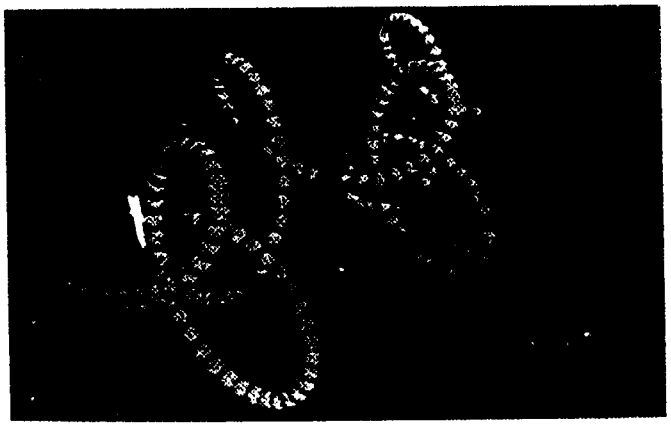


Figure 9. Graph of y=sin2x over a third dimension

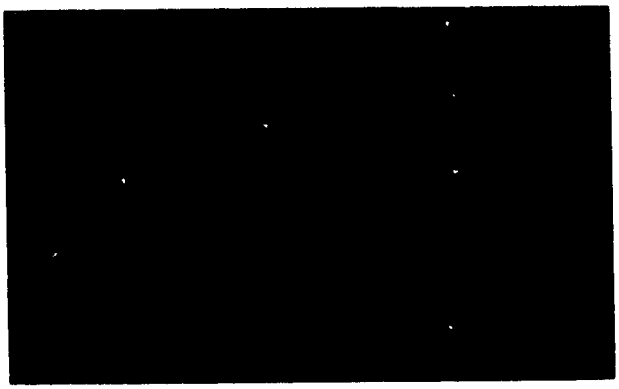


Figure 10. A height field with the elements of a julia set elevated

BEST COPY AVAILABLE

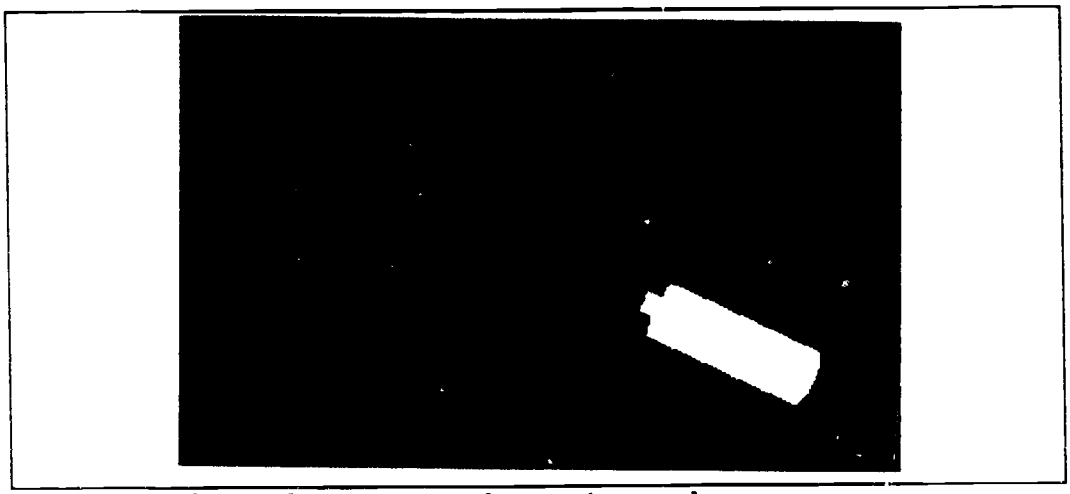


Figure 11. A flattened transparent sphere acting as a lens

al constant, or applying external magnetic and/or electrical fields, which the user can set as desired.

"Particleman", as the set-up software for the local microcomputer is named, allows the particles to be given initial masses, velocities, positions in the box and charges if desired, again set by the user. Other parameters can be varied, such as interparticle potential and final energy rate which can be used to simulate particles moving through other substances. A Particleman interface window is shown in Figure 12.

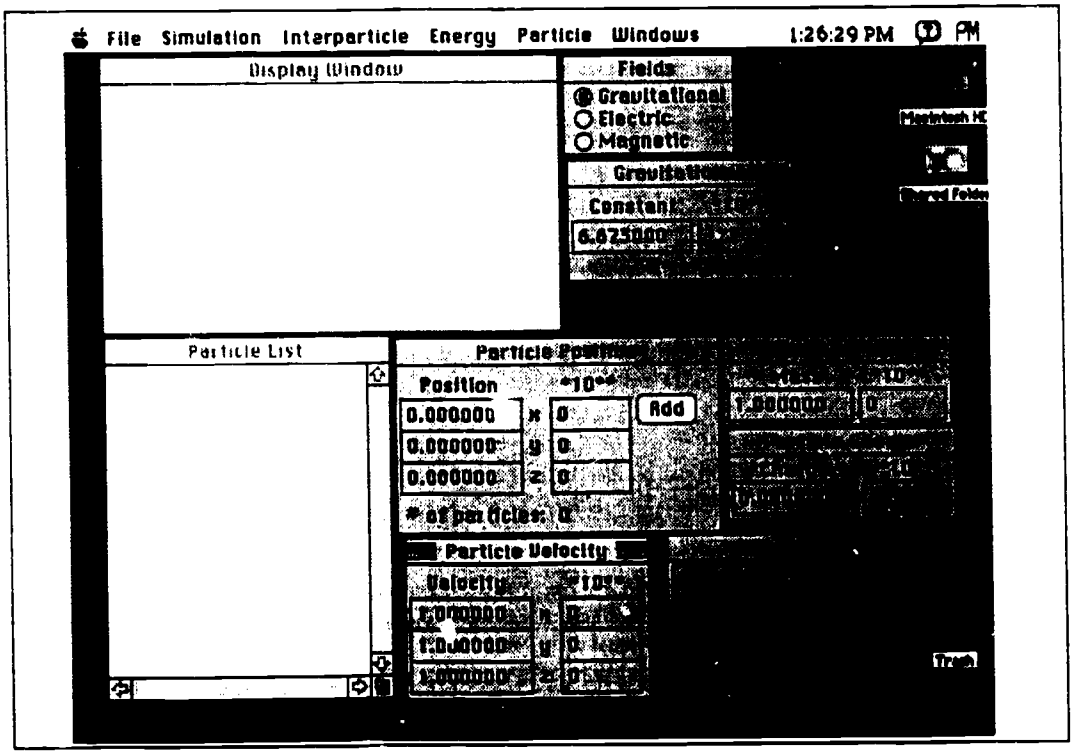


Figure 12. Particleman interface

BEST COPY AVAILABLE

One of the probiems with simulations is that computer calculations are done in discrete steps rather than continuously. The more often the calculations are done, and the smaller the time interval between, the more accurate the simulation will become of what actually would happen. Doing simulations using Particleman, the user can set how often calculations will be made, how many of them will be made, and how often a picture is to be taken for later display.

As with ray tracing, a setup file developed with Particleman is sent to the supercomputer for processing, and an output file is returned for viewing. In the case of the particle physics simulator, the viewing software is called PC3DV. The display consists of one large box, corresponding to the simulation box. This box can be rotated to view the simulation from any desired angle. There are also smaller boxes showing the orthogonal views from the x, y and z axes. When the display is run, the particles appear in the box as dots. They move according to the results of the simulation and according to the number of discrete pictures taken as specified in the simulation set-up. The particles are color-coded to show their relative velocities. The display can be run in an animation mode, where the particles appear to move in the box, or it can be run in a trace mode, in which the particles not only move, but leave a trail consisting of all their previous positions, as illustrated in Figures 13 and 14.

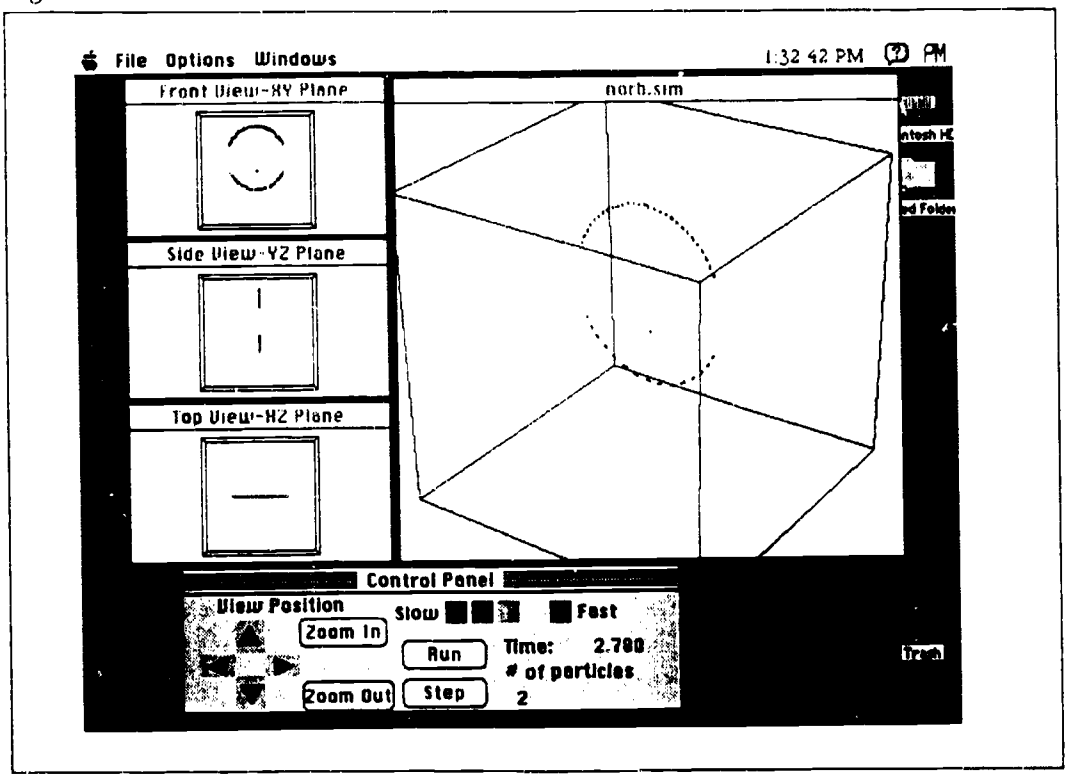


Figure 13. A simulation of an orbiting body

BEST COPY AVAILABLE

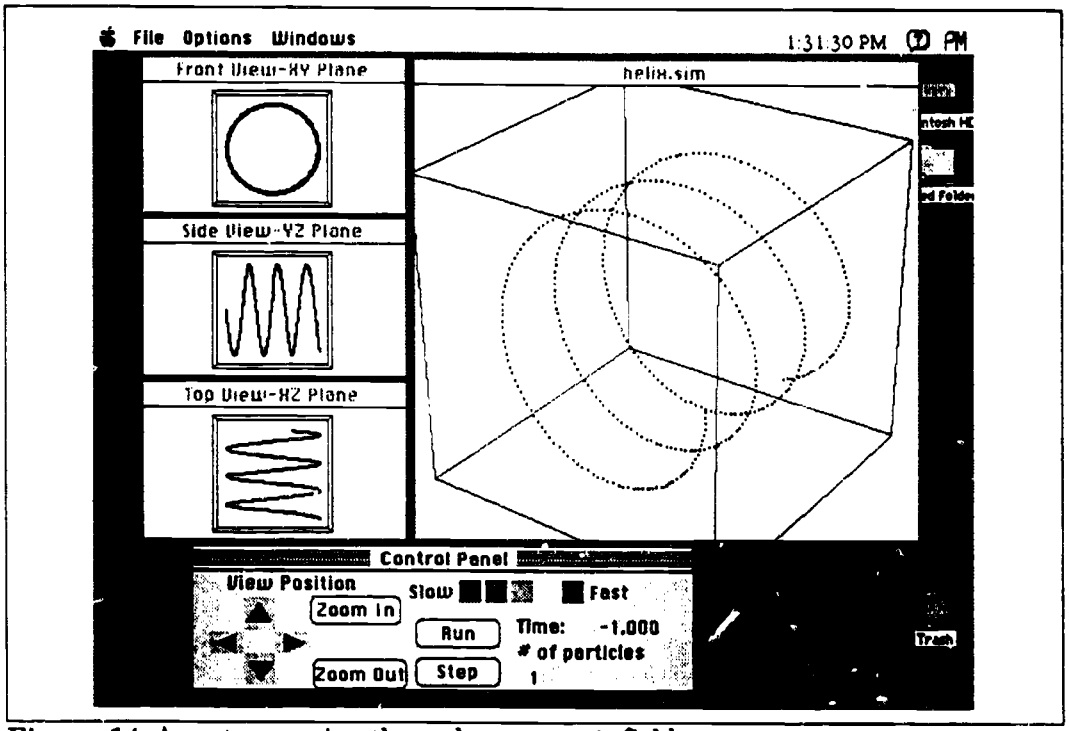


Figure 14. A proton moving through a magnetic field

More advanced users can also access a ray-tracing program generated as a result of the simulation, and alter the size and color of the particles so that the simulation is run as a ray-traced graphics movie, with spheres or other object representing the particles, complete with shading and highlighting.

The wealth of applications of Particleman in a science class are striking. Teachers can create demonstrations of concepts otherwise difficult to visualize, particularly those on very large or very small scales. Demonstrations can be interactive, since parameters can quickly be changed and a new trial processed and viewed. And, of course, it gives students a tremendously powerful tool to help them answer the "what ifs" for themselves.

Climate Modeling

Climate is defined as weather over a long period of time. Basically, climate models feature the ability to make changes to important aspects of the earth's climate system, especially changes in solar radiation, atmospheric composition, land features, and water features. Altering these features will alter the earth's climate. The climate model has the job of calculating and displaying just what those changes will be.

A climate model is based on fundamental laws of physics and consists of a set of partial differential equations. Solving these equations within reasonable time requires the capabilities of a supercomputer. The solution technique requires that the global atmosphere be subdivided into thousands of box-shaped elements. The model used in NESP is the OSU model developed at Oregon State University. For this model, the boxes are four degrees latitude by five degrees longitude.

The set-up program on the microcomputer for the NESP climate model is called Climoman. It allows for changes in a long list of parameters. Some parameters are changed numerically, including:

- solar constant—the amount of energy received on a surface oriented perpendicular to the sun's rays.
- rotation—the time of the earth's rotation around its axis.
- gravitational constant—the acceleration of gravity due to the earth.
- eccentricity—the degree of elongation of the earth's elliptical orbit about the sun.
- aphelion—the day of the year when the earth is farthest from the sun.
- radius—the earth's radius.

Other parameters are altered by using the mouse and pointer to "paint" the changes onto a world map which is displayed on the screen. These include:

- surface type—the type of vegetation in each grid box on the map. Each grid box is considered to be entirely covered with the same vegetation. The surface types available are:

- woodland/grass
  steppe and grassland
  desert
  water
  forest
  steppe desert
  tundra/mountain
  land and sea ice

As surface type selections are made and put on the map, new land masses can be formed, and existing land masses can be deleted. Students can, in effect, create their own worlds if they choose.

- geopotential—the product of gravity and the height of the land above sea level. Geopotential is used rather than height in the model because gravity is usually associated with height.
- sea surface temperature—the temperature of the water of the sea. This temperature affects the evaporation of water. Evaporation at the surface and its condensation to form precipitation in the atmosphere is a major way in which the solar radiation absorbed at the surface heats the atmosphere.
- ozone amount—the amount of ozone in the atmosphere. Ozone is a toxic form of oxygen which creates pollution and health hazards when it becomes excessive in the lower atmosphere. Higher up in the stratosphere, ozone shields the earth from ultraviolet light.

Figure 15 shows a picture of the Climoman interface.

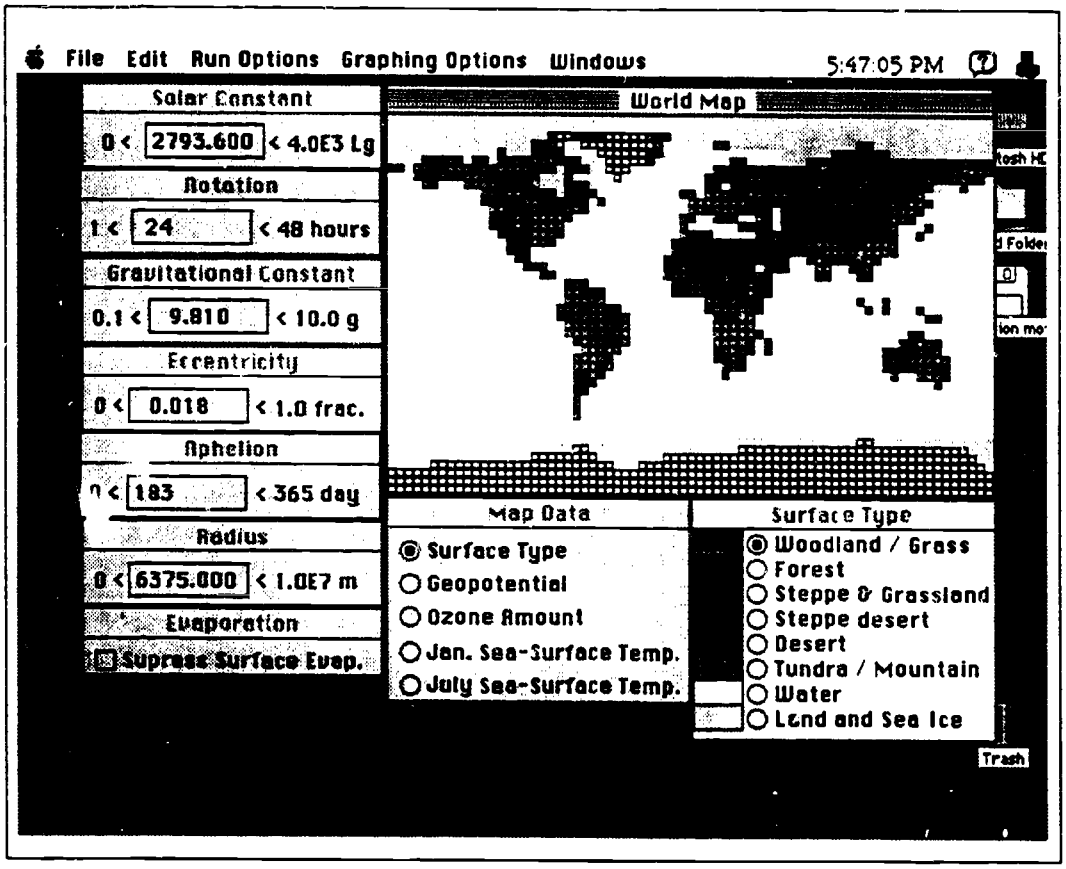


Figure 15. Climoman Interface

Output parameters are also set using Climoman. The number of days for the model to run and the starting month are established. Output can be displayed in several modes, depending on the information desired:

surface temperature	total precipitation
total cloudiness	snow mass
surface wind speed	ground wetness

Climate models are not perfect. These imperfections are called model bias. When a model is run with the parameters set to the present time's usual conditions, stated as modern standard conditions, it is often called a control simulation. Changing the modern standard conditions results in an experiment simulation. Each simulation run on the same climate model will have the same bias. Taking a difference between the control and experiment simulations will remove some of the model bias. Climoman allows for output choices of either monthly averages or differences.

As with the other applications, the Climoman setup file is sent to the supercomputer for processing and an output file returned for viewing. The climate model output file is a set of frames comprising a movie that shows a map of the world overlaid with color-coded

information according to the input parameters established during setup. When the movie is run, each frame represents a day as the climate changes moving across the map are displayed (see Color Print 6).

Classroom Experience with NESP

NESP is relatively new, so the pool of classroom experiences from which insights can be drawn is relatively limited. The approach to putting this tool in students' hands can vary greatly, as can the student target groups. NESP could be introduced as as club activity, as a unit in a science or math class, or as a class unto itself. Each approach has its advantages and disadvantages. The following is a description of our experiences at our school, Minnehaha Academy.

Minnehaha Academy is a private K-12 school with about 900 students located in Minneapolis. When we were offered the opportunity to participate in the NESP, we saw it as a chance to open some new vistas for our students, even though we at first were not exactly certain how we could integrate it into the classroom. The approach we finally took evolved as we received training.

To become familiar with the program and the software, two teachers and one student from Minnehaha attended two weeks of training at LLNL in the summer of 1991. Later that summer we conducted a two week workshop for several other Minnehaha teachers, and two other teachers from the Minneapolis area. The experiences of the summer gave us a foundation to build on.

We decided to establish a separate semester-long course at the high school level, called supercomputer applications. For the first class in the fall, the students who had the previous spring registered for an introductory computer programming and survey course were asked if they would be willing instead to participate in this new class. (Actually, the supercomputer applications became an addendum to the originally planned course.) As is true in many schools, computer equipment was limited. The class of ten students was conducted with three Macintosh computers and two phone lines, plus access to a lab with Apple II computers. As a result, the class became a hybrid. An even/odd day routine was developed for determining who had access to NESP on a given day. On their "off" days, students worked on other computer related projects, which included programming in BASIC, Logo, and Pascal, as well as some applications like Fantavision and Platinum Paint. They were also assigned several reports in the course of the semester.

For the NESP portion of the course, the early part of the semester was spent in becoming accustomed to file transfers, the utilities on the NESP Cray supercomputer (including e-mail), and the whole concept of distributed computing. As students learned the applications, they were assigned projects which involved the use of these applications. The choice of projects was left as open as possible, and students were encouraged to develop their own ideas and follow their own interests. They were given broad guidelines as to what type and how many projects were expected of them.

Additionally, students were encouraged to examine the other courses they were taking to see if there was a project which could benefit from one of the NESP supercomputer applications, for example in math, science, or even art. The hope was that the teaching and learning possibilities available through NESP would inspire other teachers and students to use NESP. An example of the use of the supercomputer tools in a science class is the student who used ray-tracing to develop a cell model for a biology assignment. Rather than

making a poster or a hand-held model of a cell, the student modeled the cell using Wireman and created a movie which focused in on the cell and then "flew" around it, showing the cellular structure from different angles. He also used scripting to label the parts of the cell. A frame from this sequence is shown in Figure 16.

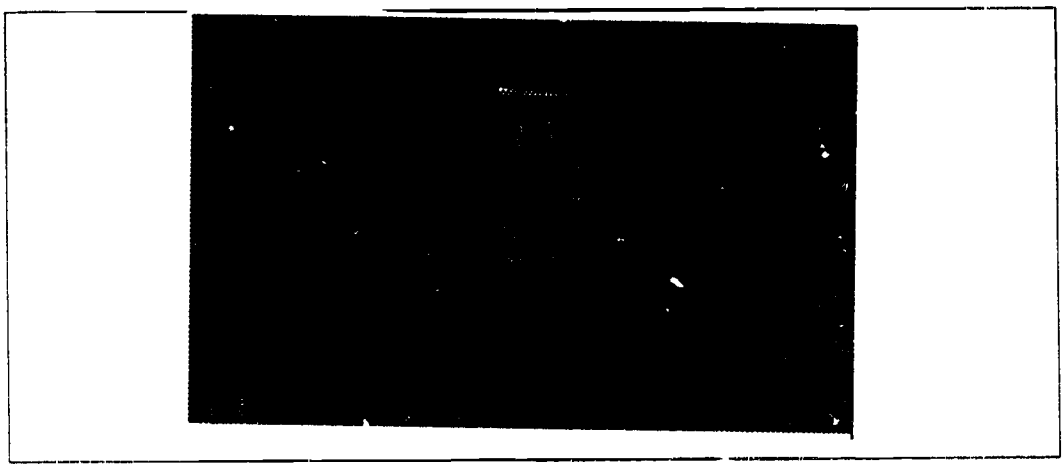
Figure 16. Cell Model (Seth Stattmiller)

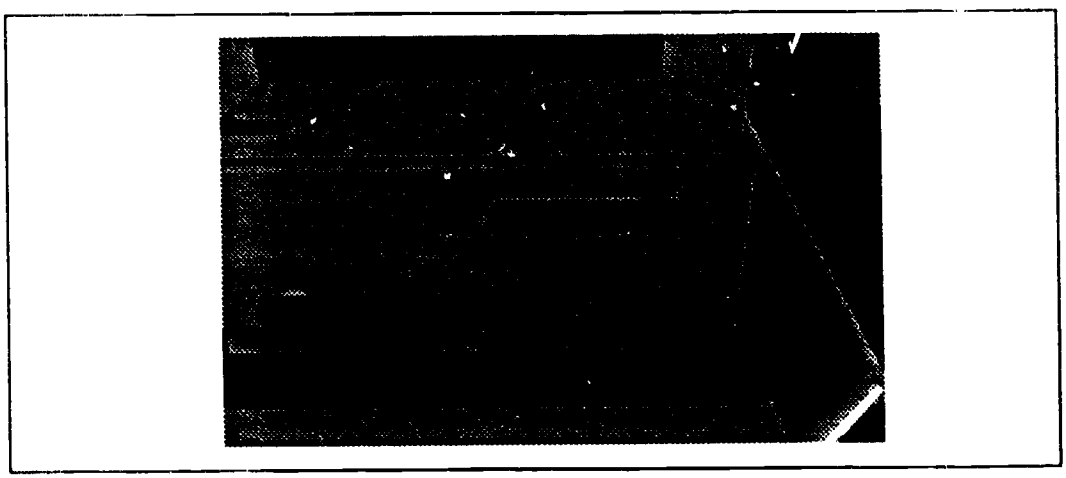
Because of time limits on access to the Macs, most student projects were not very elaborate. But the course served to whet students' appetites, and out of the ten in that first semester class, five asked if there was any possibility of continuing with a second semester. It was arranged for those five to take a second semester as a project-oriented independent study course (in addition to a new group of students in the introductory course). Each of them had an hour designated for the class, all at different times, so each had access to a Macintosh and a phone line for essentially a full class hour every day. Progress was monitored periodic personal contact with the instructor, via e-mail, and via electronic file transfer. It was with these five advanced students that the most exciting features of NESP have surfaced.

The advanced students were assigned four extensive projects. First, they were to use ray-tracing to do an "inside-out" project, in which they were to demonstrate the inner workings of any everyday object, such as a spray bottle or a ball-point pen. For example, two of the students created a movie in which they "flew" into a Macintosh via the disk drive, as illustrated in Figure 17. Another student did a project with a spray bottle, demonstrated in Figure 18.

The second project was one to be done in conjunction with another teacher. It could be to develop a demonstration, for example of conic sections shown using ray-tracing, that the teacher could then use in the classroom. Or it could be a science experiment corresponding to work being done in a science class. In one case, two students worked together with the physics teacher in developing a simulation of Rutherford's experiment. Another project was done for an art competition.

For the third project, the students were asked to do a collaborative effort with another student in another part of the country. This required the students to use bulletin boards and e-mail as well as file transfers. One student worked on a project involving the modeling of the solar system, the introduction of a "nemesis" object, and then a climate model done on the earth according to its new orbit. She worked with a Japanese exchange student living in California.





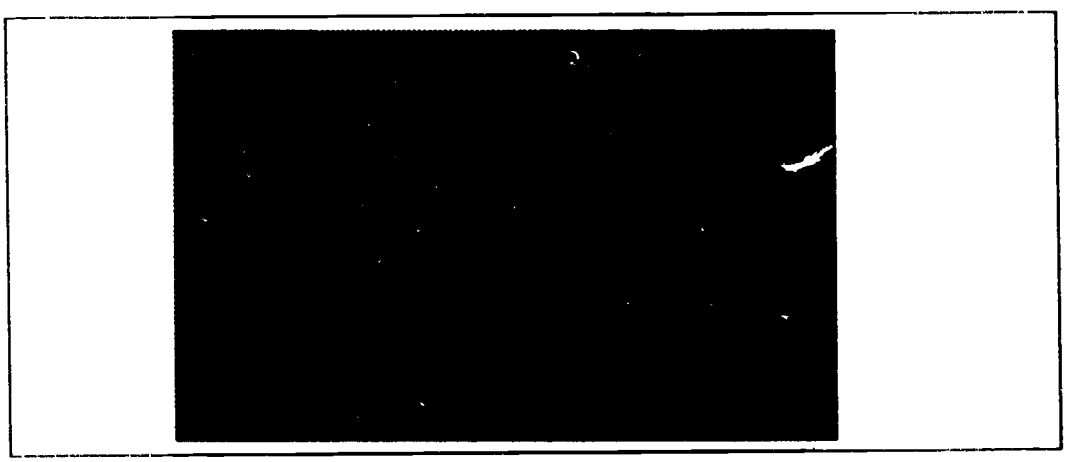


Figure 17. Three frames from the movie "Into the Mac" (Greg Anderson and Jeff Oegema)

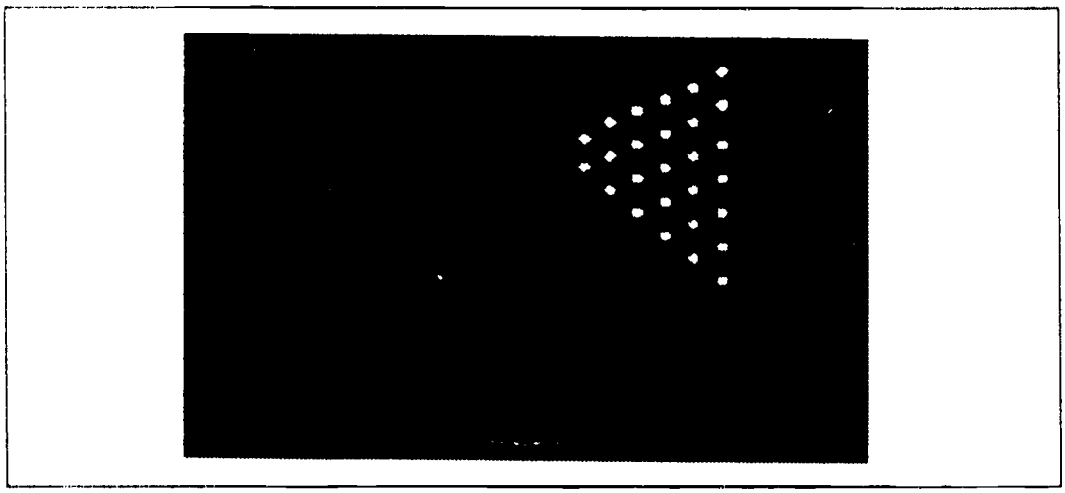


Figure 18. Three frames from the "inside-out" movie, "Spray Bottle" (April Neighborgall)

BEST COPY AVAILABLE

The fourth project was left to the students to design, but it had to involve the applications in a way that ensured that each available NESP application had been used at least once in the course of the four projects.

It was exciting to see the projects evolve. It was also rewarding to notice how classmates looked over the shoulders of these students and developed an interest in what was being done, since a major goal of the projects was to get more teachers and other students involved in and aware of the opportunities in the computer lab. The collaborative projects also helped the supercomputer students develop their communication skills.

Summary

NESP opens a wealth of possibilities for the math and science teaching. It allows for visualization of concepts previously difficult to show graphically. Students are captured by the ability to be able to build models using ray tracing, at the same time applying principles of mathematics. Experiments which are impossible to do in the high school classroom can be simulated in a very short time, and the results displayed in easy to understand movie frames. Just as important, students are exposed to the world of electronic communications and begin to learn skills which will be vital in the workplace of the future.

Participation in the program can be achieved even with limited equipment. Ideally, the school should have a lab of IBMs and/or Macintoshes, and at least one phone line with modem, but with some imagination, even a very limited supply of equipment can be used advantageously.

For a school to become part of NESP, it is necessary for at least one local teacher to attend a teacher training workshop. At the completion of the workshop, that teacher can establish accounts for other teachers and students at the school. Again, information on workshops can be obtained from Brian Lindow.

Becoming involved in the program takes some effort and time commitment, but if we are interested in providing our students with the experiences and tools to become the scientists and mathematicians and productive citizens of the future, we owe it to them to make the effort. The interest and excitement of the students and the opportunity to give them some unique experiences makes it all worthwhile.

There's an old adage:

If I hear, I forget;
If I see, I remember;
If I do, I understand.

NESP provides an outstanding opportunity to "do", not just for a few students, but for a wide range of teachers and students with a wide range of applications.

Chapter 10

New Mexico High School Supercomputing Challenge

MARILYN S. FOSTER

Scientific computing and visualization can be a real boon to education; however, providing the opportunity for hands-on experience to high school students is beyond the financial reach of most school districts and the expertise of their personnel. Community support in the form of funding and personal involvement is needed to provide a program enabling students to experience the excitement of doing science using the powerful technologies available today. The New Mexico High School Supercomputing Challenge is such a program; it helps to fill the gap between existing technology and what is available in the schools.

Opportunity is the hallmark of the New Mexico High School Supercomputing Challenge. The program provides a chance for students and teachers to discover how science can be used to solve some of the complex problems that face the world today. It is also an opportunity for the scientific, academic and business communities to roll up their sleeves and become actively involved with the scientists and engineers of tomorrow.

The Supercomputing Challenge is an academic-year-long program that gives high school students throughout New Mexico the chance to do computational science projects using high-performance computers. The program was first conducted during the 1990-91 academic year, and students, teachers, and sponsors took part with enthusiasm. The anticipated turn out for the first year was 10 to 15 teams, but the actual response topped 60 teams.

Participating schools form teams of one to five students with a sponsoring teacher and a technical coach from academia or a research laboratory. Each team defines and works on a single computational project of its own choosing. Projects from all areas of science and mathematics are undertaken with many teams choosing problems that have direct impact on their local environment. This past year, projects included sulfur dioxide pollution from a nearby smelter, a model of rangeland with and without ranching, the path of pollution particles in the state's atmosphere, dispersal of contaminants in a local water system, and congressional redistricting of the state.

The Challenge is open to all students in grades 9 through 12 on a nonselective basis. Participants come from public, private, and parochial schools in all areas of New Mexico. In the first year of the program, 235 students on 65 teams with 55 teachers at 40 schools participated, and in the second year the numbers increased to 419 students, 91 teachers, 112 teams, and 91 schools. Ten judges and more than 70 coaches volunteered their time and scientific knowledge to provide a rewarding experience for the teachers and students.

The Challenge is both an educational program and a competition. While increased knowledge is the primary goal, those teams who have made significant progress with their projects can enter them in the competitive category and vie for scholarships and savings bonds for individual team members and computing equipment for their schools. Those teams who have concentrated on the learning experience can opt not to enter the competition. Both categories of participants benefit from the educational materials and training provided.

Goals

The goals of the New Mexico High School Supercomputing Challenge include the following:

- Increase science and computing knowledge at the high school level and expose large numbers of students and teachers to computational subjects and experiences that they might otherwise not have.
- Promote careers in science and engineering by instilling enthusiasm for science in high school students, their families, and their communities.
- Encourage students to compete academically and give them the experience and confidence to enter national competitions.
- Develop a sense of team work in tackling a scientific problem.
- Reduce the isolation of teachers in remote areas by putting them in touch electronically with their colleagues at other schools so they can exchange ideas and information.
- Take advantage of existing science and computing expertise and resources within New Mexico for the benefit of high school teachers and students.

Features of the Challenge

A number of the characteristics of the Challenge have contributed to its success. The Challenge offers nonselective participation, makes available a variety of computer architectures, provides ongoing support to participants, and draws on a broad base of community support to provide the program at no cost to the schools or participants. The cost of the Challenge, discussed in more depth below, is wholly borne by the industry, government, and academic sponsors.

Participation on a Nonselective Basis

The Challenge is open to students enrolled at any New Mexico high school (grades 9 through 12). There are no prerequisites of academic classes, grade point average, or

computer experience. Those who enter the program need only be interested in learning about computing. It reaches students at schools where computing equipment and computing courses may not be available, as well as schools that have well equipped computer laboratories and advanced-level computing courses. The program lends terminals and modems to schools that need them.

There is a wide variation of knowledge among participating students--from students who do not know any programming language to those who have broad and diverse knowledge and experience in the use of computers. One student wrote, "Before beginning the Challenge, I could have been classified as virtually computer illiterate. In the end, I could understand a number of the things that computers are capable of doing and the things that I could do with them."

Because the Challenge is open to all students on a nonselective basis, the emphasis is on achievement, or competition at each team's own level. The teams of students come to the Challenge with a wide variety of starting points in terms of computing knowledge and experience. Figure 1 illustrates the differing starting points for a group of teams and the progress each made during the program. The goal of the Challenge is to maximize the learning line for each team regardless of the starting point. Real success can be measured by knowledge acquired even if a team does not reach the point of having a competitive project.

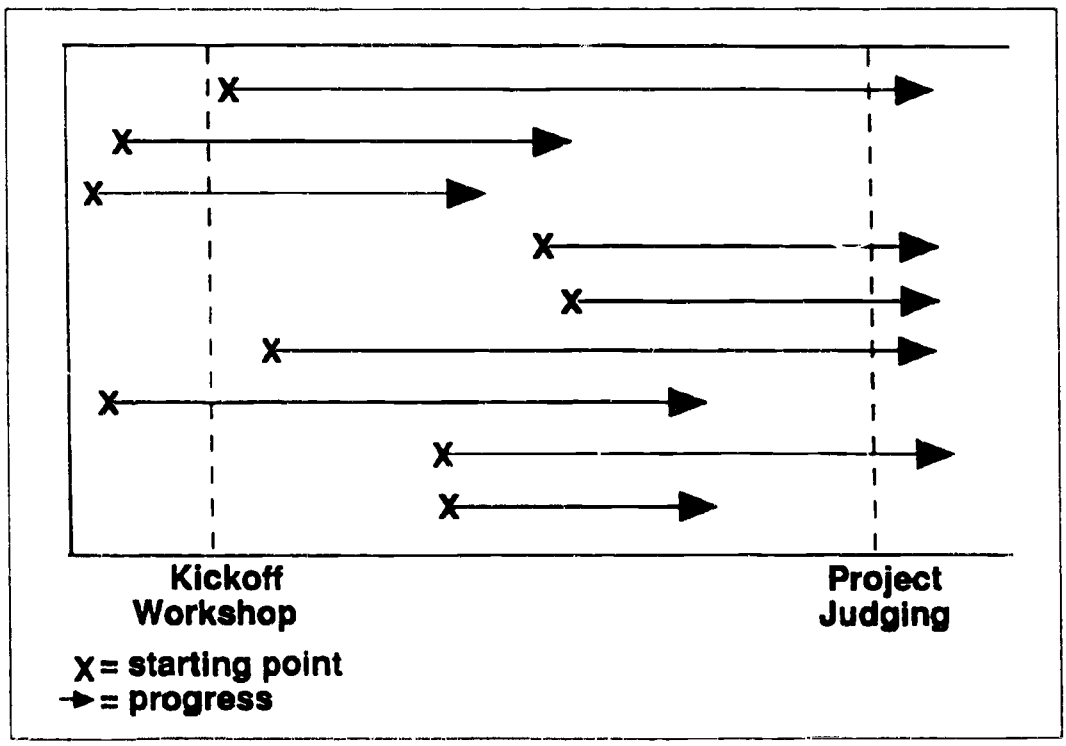


Figure 1. This diagram shows the variety of levels of computing knowledge and experience of teams beginning the Challenge, and the progress made during the program.

The program reaches many students who are historically under represented in scientific fields. In the initial program in 1990-91, women represented 25% of the students and 35% of the teachers, and three teams were all women. By the second year, the number of women students had increased to 30% and the teachers to 43%.

REGISTRATION STATISTICS

	Challenge 2	Challenge 1
Schools	51	41
Teams	112	65
Students	419	231
Male	70%	74%
Female	30%	26%
Hispanic	38%	22%
Native American	11%	3%
Asian	3%	2%
Black	2%	1%
Teachers	91	60
Male	57%	67%
Female	43%	20%

Figure 2. This chart shows the number of participants in various categories for the first two years of the program. We hope to increase participation among minority students by seeking the support of organizations that promote awareness and academic involvement for these groups.

In Challenge 1, approximately 75% of the students were seniors, with most of the others in the junior class. The second year of the program saw a broader distribution with 43% in the 12th grade; 36%, 11th grade; 15%, 10th grade; and 6% in the 9th grade. Many teachers remarked that they were encouraging students to start entering the Challenge while in the earlier grades so they could gain the knowledge and experience needed to be competitive by their junior or senior year.

Schools may join together to form a team. Students from the New Mexico School for the Visually Handicapped joined with sighted students from a local high school to form a team—both groups benefited from the arrangement.

Access to Different Computer Architectures and National Networks

Students are provided time on CRAY, Connection Machine, Convex, IBM, and VAX computers, where they may explore the different architectures in finding solutions to their problems. All participants are given accounts on the computers at Los Alamos National Laboratory, and teams may also request accounts on machines at Phillips Laboratory and Sandia National Laboratories. More than 125 hours of supercomputer time were used by participants during the first year, and over 300 hours were used during the second year.

New Mexico Technet provides state-wide computer network access to these computers via 1-800 and local telephone lines. As the number of participants increases, it has been necessary to add several more telephone lines to this network to accommodate the usage. One student moved to a school 150 miles away at midyear. This school was not participating

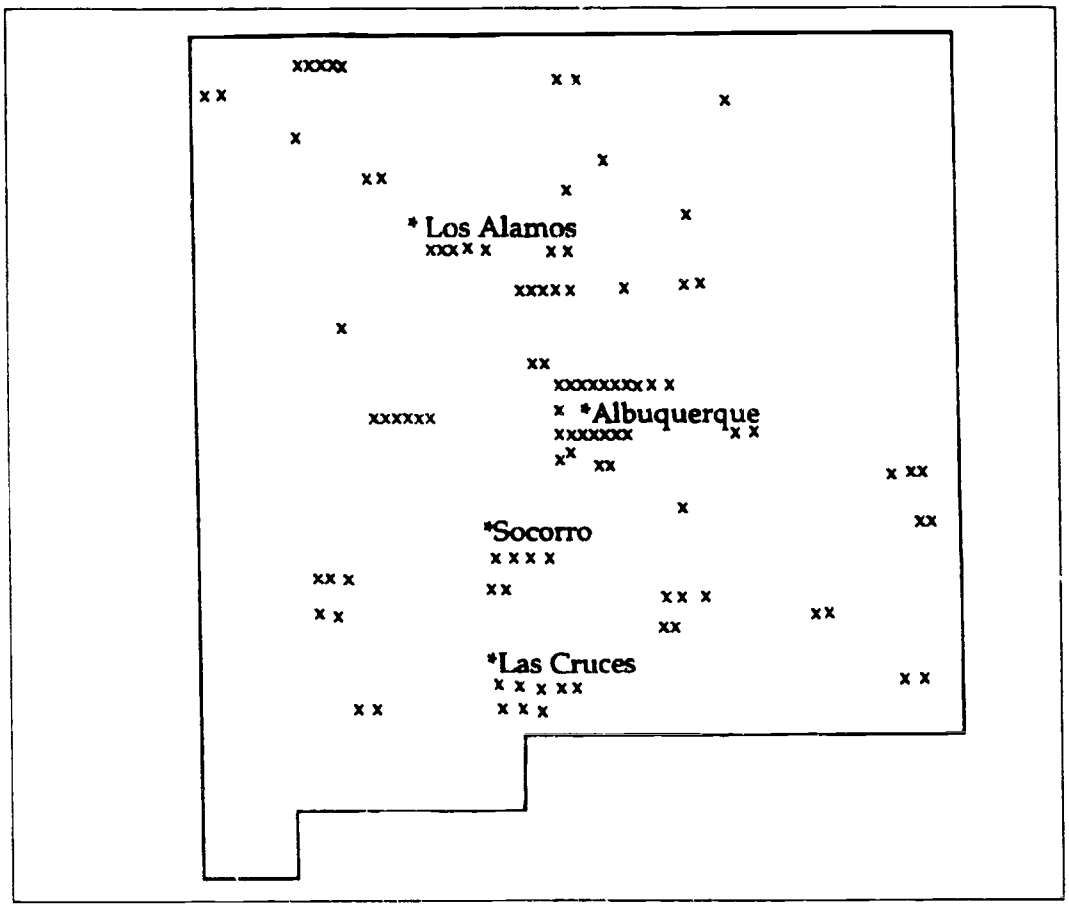


Figure 3. On this map of New Mexico, you can see that the teams are located all over the state, far from the sponsoring institutions in the major cities. Over half of the participants are from small towns and rural areas.

in the Challenge, but through the electronic network, the student was able to continue working with his team and complete the project.

Access is also provided to the national networks. By using the internet, students can access information from around the world and contact experts working in their scientific area.

Ongoing Support

Throughout the Challenge, a wide variety of support is provided to the participants. Each team receives

- training and computer documentation;
- equipment loans if needed;
- communications and computing troubleshooting and consulting;
- scientific coaching.

Training: The Challenge begins with a kickoff workshop in October where scientists talk about supercomputing and its application in different scientific fields. Instructors conduct hands-on laboratory sessions during which participants log on to the computers through the communications network and use basic commands to compile and run a sample program. In addition, district workshops are held during the course of the year at five locations around the state to answer questions and provide further instruction.

Documentation: Each participant receives a notebook of documentation, which gives them quick tips about using electronic mail, file editors, file storage, and basic UNICOS commands. Online documentation is available on many topics, and teams may borrow reference material for additional help. Each team receives a reference package of programming texts and information about graphics programs and output. The team reference information remains in the school so that each participating school can build a library of computing information.

Equipment: Because many schools do not have adequate equipment to access the network and supercomputers, terminals or workstations and modems are lent to those schools. An equipment and communications technician contacts each teacher to determine what is needed, finds the appropriate equipment, ships it to the school, and remains in touch to be sure that everything is functioning properly.

During the second year of the program, to make it easier for students to include visualization in their projects, district graphics centers were established at a number of locations in the state—primarily at college or university campuses. At the centers, students in the Challenge have access to high quality graphics workstations on which to view their output. A team can go to a nearby center, transfer their files from the supercomputer to the workstation, and either write a program or use existing graphics software to view the output. The centers also contain extensive reference documentation not available in the local schools, and each center has a local consultant available to help the students.

Computer and Communications Consulting: Throughout the year, computer consultants at the supercomputing centers are available to answer questions and solve programming problems, and personnel at New Mexico Technet handle communications questions and problems. Questions range over a wide variety of topics, from "Why can't my teammate log on?" to "How do I send my files to the Connection Machine?"

To call for help, most of the participants would have to use long-distance telephone service, so they have learned to use electronic mail instead. They also use electronic mail to get to know each other and share information about computing and their science topics. Teachers share experiences and information with their peers and come to realize that they are not the only one encountering a particular problem.

Technical Coach: Teams are assigned technical coaches from either academia, a scientific research laboratory, computer vendor or other source. The coach is someone familiar with supercomputing and software development and is an expert in the scientific area of the project. These people help guide the team members in selecting a do-able project, direct them toward available resources and information, provide technical information

about the science and math required for the project, and help them select appropriate software and the proper computing platform for the problem. The relationship between a coach and team can be rewarding to everyone as the coach assumes the role of a mentor to the individual team members. As one of last year's teacher expressed it, "The coach's role is to debug the team."

Broad Base of Community Support

The Challenge is a cooperative effort sponsored by a partnership of universities, national laboratories, businesses, and New Mexico Technet. Representatives of these organizations oversee the general operation of the Challenge and raise funds to support it. The day-to-day functions of the program are carried out by personnel from Los Alamos National Laboratory, Phillips Laboratory, New Mexico Technet, and the University of New Mexico.

Implementation

The Challenge is divided into six phases during the academic year. Feedback from participants as well as the organizers is continuously gathered and evaluated. Adjustments are made to the program as needed.

Phase 1. Call for Participation: The Challenge begins at the start of the academic year with a call for participation. Teachers and students form teams and enter the Challenge by submitting an application form listing the team members and describing, in general terms, the scientific area and project they plan to work on.

Phase 2. Introductory Workshop: In October, all participating students and teachers are invited to a two-day workshop where they are introduced to computational science and learn the basic skills needed to work on a supercomputer. The workshop includes hands-on laboratories where students and teachers sign on to the supercomputer just as they will do from their schools.

The atmosphere at the workshop is both enthusiastic and intense. At the first opening workshop, a dance band played to an empty room while the students went to the computing lab. At the second workshop, instead of a dance, extra lab sessions were planned for the evening, and the students kept the rooms full until after midnight.

Phase 3. Initial Work: After the kickoff workshop, students return to their schools to begin tackling their problem. Many students must also learn a scientific programming language such as Fortran or C. If a programming class is not part of their school's curriculum, some students take a text book and learn it on their own. One teacher noted, "Students learn much on their own and learn faster because they see a relevant need."

Phase 4. Computing: The computing phase of the project is the most exciting and frustrating to the students. The consultants are in great demand at this time answering programming questions and examining problems.

In the second year of the Challenge, two district workshops were added to provide better support to the participants during the computing phase of the program. A team of experts from the sponsoring organizations travels to five regional sites at colleges or universities in the state and spends a full day at each location meeting with the teams from that area to answer questions, give project and programming hints, and teach skills that the students may need to complete their projects.

Phase 5. Final Report: Teams must complete their projects by mid-April, write a final report, and create a poster display. The reports are submitted to the panel of judges who read them and decide on a group of finalists. The members of the finalist teams meet with the judges to present their projects and answer questions covering both the scientific content of the project and the knowledge of the team members. The judges want to be sure that a project is the work of the students, not that of the teacher or coach.

The posters describing the projects are displayed at the Awards Day in late April. The teams are always anxious to see what teams from rival schools have done. Ribbons are presented for the best posters as determined by the viewers, and the poster receiving the highest number of votes is incorporated into the design for the Challenge logo for the following year. The logo appears on a tee shirt given to all participants at the kickoff workshop.

To accommodate the diverse computing background of the students entering the competition, projects may be submitted in either of two categories: competitive and noncompetitive. Category 1 is the competitive track for teams who have made significant progress and choose to submit their projects for judging. Category 2 is for teams who have made progress in learning to work on a computer but were not able to make significant progress on their problem. These reports include what the team members have learned about the scientific area of the project and what they have learned about supercomputers, programming, and communications.

Phase 6. Judging and Awards: Leading scientific and computing experts drawn from research laboratories and universities in the region are asked to participate as judges. They read the reports submitted in Category 1, choose the group of finalist teams who must give short presentations describing their projects and answer questions from the judges. The projects are rated according to the following criteria:

Scientific content	30 points
Effectiveness of approach ..	30 points
Creativity	30 points
Clarity	10 points
Total	100 points

At the culmination of the competition, all teams (whether in the competitive or noncompetitive categories) are invited to attend a one-day awards ceremony and science tour at Los Alamos National Laboratory, where they can see the supercomputers they have been using and talk to scientists who use computing in their daily activities.

Awards for Category 1 teams in each of the first two competitions included: first-place awards of a $1,000 savings bond for each student on the team and a fully equipped work station for the school; second-place awards of a $500 savings bond for each student on the team and a personal computer for the school. Each member of a team that completes the Challenge by submitting a report in Category 1 or 2 receives a momento of their participation in the program.

In addition to team excellence, individual excellence and leadership is also rewarded with three $2000 scholarships to state universities. These awards are based on a report written by the student to describe his/her role on the team and substantiated by letters of recommendation from the teacher and technical coach.

Phase 7. Evaluation and Feedback: To monitor the effectiveness of the program and meet the needs of teachers, students, and coaches, input from all participants is continually gathered. Feedback forms are used at all workshops, the quality of the completed projects is evaluated, and computer usage time is observed.

Results

In the initial two years of the program, the Challenge has achieved a high level of success. The extraordinary interest among teachers and students from throughout the state was a surprise, and the organizers were delighted that 80% of the original participants in the first year completed the Challenge. This level of interest indicates that the Challenge is a worthwhile program, and that high school students are interested in learning about and working on supercomputers.

The students and teachers profited from the program in many ways.

- They learned about computing and science.
- Horizons were expanded.
- Science and technology have a greater place in their lives.
- Computing is viewed as vital in science and the classroom.
- Students have a new interest in science and computing as a career.

Additional observations after the first year of the program include the following:

- The goals of promoting interest in science and computing and increasing students' knowledge in these areas can be achieved through this type of program.
- Despite frustrations and severe time constraints, students will persevere in an exciting program; the Challenge had a better than 80% completion rate.
- Ongoing support is critical to identifying hurdles and addressing them rapidly. Teachers have little time to spend on troubleshooting and seeking solutions to problems.
- Technical coaches are necessary to give direction to the projects and individualized support in science and computing. Many teachers need additional science and computing knowledge and experience, and coaches are a ready source of relevant information.
- A nonselective program is key to achieving broad participation leading to broad benefits. The Challenge is not exclusive in terms of academic achievement or class ranking.

- Lack of additional teacher training holds teachers back. They must rely greatly on the technical coaches. We recognize the need to conduct inservice workshops during the program, but lack of staff and funds has prohibited it.
- Lack of equipment, telephone lines at schools, computing knowledge, and teacher and student time have been frustrating to everyone. But continuing support has been effective in quickly identifying and addressing these problems.

Challenge Projects

The problems that the Challenge teams choose to solve during the program show the diversity and ingenuity of teenagers. While fractals, neural networks, and astrophysics are popular topics, some teams are more interested in finding the optimal design for the parking lot at the football stadium, discovering the ideal fuel to air ratio in an automobile engine, or examining the predator/prey population in their local area. The titles of some of the award winning projects show this diversity:

The Problem with Rings: A Simulation of the Rings of Saturn
First Place, 1991
Fractal Dimension: Theory and Computation
Second Place (tie), 1991
Behavior of Ions in Zeolites
Second Place (tie), 1991
Examining the Evolutionary Tendencies of a Computer
Simulated Organism with the Aid of a Genetic Algorithm
First Place, 1992
Thermal Topography: Heat Distribution in Malignant Tumors in Human Tissue
Second Place, 1992
The Simulation of Fractal Growth Using the Diffusion-Limited Aggregation Process
Honorable Mention, 1992
The Congressional Redistricting of New Mexico
Honorable Mention, 1992

Some of these projects used visualization to display the results. Many of the teams had planned to incorporate graphical representation into their project, but were hindered by lack of equipment, time, and expertise. As a project in a video production class, one member of the 1992 winning team is trying to visually display the results of the team's work. Enabling the teams to do more graphical representation of their projects is one of the ambitions of the Challenge, and plans are being developed to achieve this.

Future Plans

The New Mexico High School Supercomputing Challenge is a dynamic program undergoing continuous evaluation and change. As the program grows, adaptations and improvements will be made to meet the needs of the participants and encourage a high level of academic excellence. Wide spread participation especially among schools in disadvantaged areas will be promoted.

To overcome the obstacles teams have in using visualization in their projects, the Challenge has established five regional graphics centers located at colleges or universities around the state. The teams can go to the centers and use workstations to generate graphics programs. Each center has a resident consultant who can help the students. Equipment will be upgraded and the resident staff will be instructed to more effectively work with the high school students and teachers.

Attention will be directed toward teacher education with additional training in scientific computing and the use of supercomputers in science provided. Instruction in the use and techniques of visualization will enable teachers to employ it as a teaching tool in all areas of science. Professional development programs being considered include a summer institute in computational science and several regional one-day in-service workshops.

The State of New Mexico, in conjunction with U.S. West Communications and New Mexico Technet, is developing plans to enhance the communications network so the students at their high schools can have direct network access, instead of limited telephone access.

Summary

The New Mexico Supercomputing Challenge is presented as a link between the education communities and the science communities, both locally and nationally. It enhances the knowledge of high school teachers and students in science and computing; it increases scientists' understanding of educational realities and opportunities; and it taps the potential for community benefits offered by scientists and the energy and desire to learn of teachers and students.

The Challenge is an effective way of encouraging students to discover the excitement of science and computing. The enthusiasm generated by academic success can provide the impetus to pursue a career in science and computing. By combining science and computing with a high school activity of high visibility and excitement, students, parents, and teachers are attracted and will learn more about scientific computation and the multidisciplinary approach to scientific problems.

The Challenge may serve as a model that can be enhanced and adapted by other localities. A nationwide scheme of local programs leading to the national level could increase the level of interest and the number of students participating in scientific and computing endeavors and foster higher levels of academic achievement.

Without the influx of significant science and computing expertise and resources into high school educational communities, high school teachers cannot by themselves enhance science education. The Challenge uses scientific and computational expertise existing in research laboratories and universities, in collaboration with education communities, to make a difference in high school science and mathematics education.

If you would like further information about the New Mexico High School Supercomputing Challenge, please contact

New Mexico Technet
4100 Osuna, NE, Suite 103
Albuquerque, NM 87103
(505) 345-6555

You may also phone Patricia Eker, Los Alamos National Laboratory, at (505) 667-3193.

Chapter 11

Sharing Multiple Complementary Representations in the Teaching of Science

NORA H. SABELLI
IGOR S. LIVSHITS

In this paper we will address a number of issues related to the use of visualizations in science education from the perspective of a scientist working with visualizations to teach his or her discipline. Some of the issues raised by this perspective help frame the work with students that will be described; the rest are contributions to the research dialog we wish to establish:

- Multiple representations
- Role of tools in science
- Visualization of simulations vs. animations
- Simplifications vs. complexity of real phenomena
- Modes of learning

The vision that underlies our work is the power of the methodology enabled by advances of high-performance computing and communications (HPCC) to transform the construction of knowledge and the development of scientific intuition, be it in scientists or in students entering into a complex new field. In particular, HPCC methods and techniques may help to:

- Empower teachers and students through access to desktop visualization and networked resources;
- Change the current paradigm of teacher as expert and student as receptor;
- Allow students and teachers to learn by discovery;
- Allow more and more diverse students to enjoy, study, and understand science and mathematics (Sabelli, 1991).

ERIC

Chemistry as a Case Study

In contrast with physics and biology (and even mathematics) education, where mechanistic simulations have been extensively used and studied, chemistry cannot draw on students' previous intuition to aid in their understanding. As children we all learned over a long time the intuitive skills needed to count, measure, gauge speed and force, and understand motion, birth, death, and growth, among other concepts. These concepts (and associated misconceptions) are macroscopic in nature and therefore observable (Larkin, McDermott, Simon, & Simon, 1983); this is not true of many of the basic concepts of modern chemistry as it is practiced and taught now, even at the introductory level. When we see chemical reactions (cooking, moving cars, sun-burning, healing) we do not see what we are taught in chemistry and therefore, often enough, we do not see "chemistry" at all. This has led to an increased use of "cartoons" and "animations" in chemistry education in an attempt to help students create mental models.

Since the introduction of an "atomic" theory to explain why substances combine in fixed, constant ratios, the interpretative framework of chemistry has been microscopic. The electron-pair and other microscopic models that chemists have developed to interpret and predict reactions have survived, and are indeed embedded, in computational chemistry. A beautiful example of the merging of classical and quantum chemistry, and of their survival as valid representations to this day, appears in Robert Mulliken's (1966) Chemistry Nobel Prize address. Coupling simulations with visualizations (see Color Prints 7 and 8) presents chemists with a powerful description that matches the three dimensional, dynamic nature of their field; computational chemistry results are placed in the context of chemists' intuitive models.

Molecules have reality; mathematical functions (wave functions or orbitals) represent the state of electrons in a molecule, but do not have "reality" in the same sense. Wave functions are written as mathematical expressions, which have value at every point in space. We extract from these molecular wave functions model constructs to a) think about molecules with concepts derived by chemists over many decades, and b) derive calculation procedures to predict and interpret observations. We, chemists, switch as easily between the complementary models of orbital wave functions and electron orbits, as we do between the models of delocalized and localized orbitals. We choose a model to use based on the question we pose: if we ask about reactions, we think of hybrid localized orbitals and picture electron jumps from one atom to another, and of bonds breaking or forming (valence-bond picture). If we ask about spectra, or other molecular properties, we think of delocalized molecular orbitals and excitation of electrons in the molecule (molecular orbital picture).

Chemical intuition does not come easily; students have many problems understanding dynamic, three-dimensional processes. For example, success in handling equilibrium as chemical algebra has been found to mask an underlying lack of chemical understanding. Students with better understanding of chemistry may commit more errors than novices who ignore underlying concepts while doing the algebra (Hesse & Anderson, 1988; Kozma, Russell, Johnston, & Dersheimer, 1990). Success in studying organic chemistry has been correlated also with the ability to rotate mental images (Prybil & Bodner, 1987), a skill found to be gender-specific and acquired with training.

The terminology and basic toolkit of quantum chemistry (for example, atomic and molecular orbitals, hybridization, localization) is now part of chemistry discourse; energetics of reactions are routinely calculated and analyzed by chemists in all sub-specialties of the field. The practice of chemistry has been marked by close collaboration between

experiment and theory; chemistry is a discipline that makes very effective use of available computing resources in simulating the behavior of complex systems. It is in applied industrial applications (pharmaceutical drug design, biopolymers, catalysis and materials science, for example) that the importance of such use becomes clear.

Teaching Difficult Concepts

Hybridization is introduced in textbooks as a fact whose details (sp; sp^2; sp^3) are fixed. But hybrid orbitals are only a useful shorthand to represent in a static, diagrammatic form the energy-controlled continuous three dimensional evolution of atomic orbitals upon formation of molecular orbitals in the presence of other atoms. The model "hybrid orbital" is useful for thinking and communicating a more complex picture, one that uses many mental constructs that chemists are familiar with.

There is no doubt that students need to understand both complementary views at some point: chemists do not have a single model. Can we use visualization to make this process easier or more intuitive? With current technology, students now may see bond formation happen as a continuous change in density, see the energy of the combined system drop and then rise as the distance between atoms becomes smaller, even see the changes in potential and kinetic energy as the bond forms and see how one extracts from the integrated picture different models to use as mental shorthand. Even better, students could see the molecule in continuous vibration and could see the molecule in three dimensions. We have used the word "see" for its meaning of "grasp the concept," as much as for its meaning of "observe."

Description of the Work with Teachers and Students

The focus of our work has been implementation; evaluation and research are starting, and is being conducted in collaboration with experts. The aims of this presentation are to encourage researchers to study the questions that our experiment and the education uses of HPCC technology have raised.

The project described is addressed to the general high school chemistry population working in Champaign-Urbana schools. Work continues under an NSF grant with four high school teachers and their classes (David Bergandine, Terry Koker, Robert Miller, Barry Rowe) and other scientists (Harrell Sellers and Kenneth Suslick). Teachers outside Champaign-Urbana have adapted the work. As the second year of work with general chemistry students is in progress, we will concentrate our remarks on the results of the first year (1991) experiment, conducted primarily with AP chemistry students. The results of current work will be published by the high school teachers working on the project. Anecdotal information indicates that our conclusions do stand up with a different student population.

Conversations with local high school teachers and other colleagues, and preliminary work to develop examples for teachers to show in class, led to the joint development by NCSA and high school chemistry teachers of a prototype computer experiment centered on the concept of ionic v. covalent bond formation. The description corresponds to work done with Mr. Barry Rowe and his class at Centennial High School in Champaign-Urbana. The exercise consisted on students producing short movies of the formation of a bond, as visualized calculated electron densities, and timed to correlate with class discussion of the topic. The molecules LiH and Li_2 were chosen to minimize computing time; various and

diverse choices are being made during the present phase of the project. The calculations were performed on NCSA's Cray Y-MP using an existing molecular orbital package GAMESS (Schmidt et al., 1990). GAMESS and similar packages run on many computers of different power and cost; which computer one ultimately uses will depend on availability of resources and on the size of the problem being studied. Current work uses NCSA DISCO developed by Harrell Sellers (Almlof, Faegri, & Korsell, 1982; Saebö & Almlof, 1987).

NCSA software tools provide a mapping of numbers into colors. The structure of the visualization is in the data itself, and the mapping may be manipulated interactively. The resulting electron densities were visualized using NCSA's Image and Datascope software on a Macintosh II; no further manipulations or animation of the densities were performed. Characteristics of the exercise follow:

- The topic was discussed in class by the teacher, and the students were encouraged to think about what concepts they did and did not accept ("believe");
- This was augmented during the laboratory with a brief, informal discussion of how calculations are made, aimed at justifying the mathematics used from the standpoint of basic physical concepts (potential and kinetic energy);
- Students worked in pairs on Macintosh IIcx workstations, where they downloaded and visualized the density maps;
- Since the visualizations were not animations or cartoons, the students were challenged to extract from them the models and concepts that had been discussed in class. With the software used, this involves manipulating color palettes to change the mapping of numbers into colors;
- Students controlled the parameters of the calculation: internuclear separation, model parameters (basis set size), mapping parameters (step size and the density map plane), visualization parameters (mapping of density values to colors in NCSA Image);
- Students chose or were assigned a molecule, or one of the orbitals of a molecule (i.e., the valence or core orbital).

The methodology can be best described as "visual interpretative experimentation." All 20 students in the class took part in the exercise. The students were interviewed and videotaped at different times during the laboratory. The most significant observations included the following:

1. Students enjoyed choosing parameters to use in the Cray calculation, and referred to their choice when interpreting visualizations; this feedback mechanism played an important part in their successful interpretation of the results;
2. Students immediately picked up changes in the visual images and concentrated their thinking on these changes. Students were able to argue whether the observation corresponded to a meaningful, interpretable condition, or was an artifact of the simulation;
3. Students followed independent exploration paths, and were able as a group to ask themselves meaningful questions (Why are the core orbitals not showing the same effect than the valence orbitals? How close should I bring the atoms to have the cores interact?);
4. Students branched easily into different systems, leading the discussion in class to additional topics that are normally part of the curriculum, but at a later date;

5. The previous questions and others led to an (unplanned) extremely illuminating discussion of the numerical and conceptual limits of the model;
6. The first team to complete a movie of their reaction was formed by the two young women in the class who "hate computers" and had serious misgivings about the process. This observation has been repeated in current laboratories.

From the standpoint of the current discourse on education reform, what strikes the authors as most interesting is the change from confusion about abstract concepts that hid the chemistry to concrete understanding of the process on the basis of the same abstract concepts. Here are excerpts from Mr. Rowe's evaluation:

> High School chemistry students have been taught various models of the atom over their school career, starting with "billiard ball" models; progressing through more sophisticated models until they are challenged by the wave-mechanical model. However, most of them accept the most sophisticated model they can still visualize. This is usually the Bohr atom....a very unsatisfactory model. For teenagers to accept a mathematical model, with probabilities and electron densities determining the atom's shape, they need more than static pictures. Static pictures, even good ones, are like an "artists' conception" of the atom. [Several students commented independently that they would not believe any drawing they thought was 'made up'] They could be one man's idea of what the atoms could be.
>
> What is very important is that the student is making the decision as to what calculations are to be made. The distance between the atoms, the energy levels modeled, and what atoms to model are made by the students doing the calculation. As they animate their pictures and make the model dynamic, they are able to explore... and determine the shape of the molecule. Before my students modeled atoms interacting, there was a genuine reluctance to give up their comfortable, particulate model of the electron and refusal to accept wave mechanics. They actually argued against quantum theory, because they could see no way that mathematics could represent something that had a shape but was not "a piece of matter." I think these students have a better understanding of atomic structure than most juniors in College.

Discussion

We believe that an important reason behind the success of the laboratory was the sense of empowerment of the teacher; this has become even more notable in the second year when four teachers work together in developing and testing materials. The teachers' increased understanding of the chemistry involved and of its theoretical basis pushed towards a growing sense of educational efficacy and enthusiasm.

Both teachers and students appreciated most the use of professional tools and the insight into how the concepts studied in class had been obtained. This increased acceptance of the concepts presented as valid working models was a powerful motivation for independent student work. For example, forcing the students to choose between spending computing resources to obtain a visually clearer picture—as in the textbook—and interpolating numerical values to refine the coarser image provided students with a sense of intellectual respect that contributed to their efforts at understanding.

We would like to emphasize that it was not the visual interest of the images that held the students' attention; it was the work they had to do to extract the orbital model described in class from the visual representation of data. It would be interesting to determine if animations would have been as successful; students enjoyed the intellectual exercise of correlating in their minds the image with the visualization. As an example, their comments made it very clear that they could believe the hybrid orbital in the book only as a step in a (more complex) dynamic transformation they had obtained, when they would not believe a cartoon of the same hybrid in the textbook. The contradiction between the students' personal mind image and the one they produced forced them to confront model representations.

What Questions Would We Like to Have Answered?

Specifically, we want to evaluate our success before the end of this project by:

- Conducting interviews with students participating in testing the modules (and who have not participated in the development of the examples) to estimate the tools' effect on attitudes towards science that may lead to changes in student career choices;
- Working with participating teachers to gather evidence of effect of the methodology on student understanding and motivation, as perceived by the teacher;
- Working with participating teachers and others to document the additional chemistry content areas that either may be changed or may be introduced earlier in the year, or that may be introduced for the first time in the experimental classroom;
- Monitoring students' performance and future career choices.

We will publish the results of the project so that the approach may be extended by others to study fundamental pedagogical questions concerning (Blumberg, Epstein, MacDonald, & Mullis, 1986):

- Novice v. expert understanding of simulations and graphical representations;
- Development of mental models of basic concepts derived from complex and real simulations;
- Effect of manipulating the models on developing understanding and intuition.

Availability of tools and prototypes of classroom materials will greatly facilitate these investigations. The project is, in a sense, an exercise in exploring the impact of expert mentoring on teaching science. This is, to the authors, the most exciting aspect of the work: the transformation and depth in understanding of chemistry by teachers and students when exposed to chemistry as it is practiced. Visualization of their own calculations with minimal interposed interpretations give students a stake in understanding, and a way to make concrete abstract concepts introduced in class. Use of these concepts in a dynamic way, and insight into the basic reasons of why models have been chosen by chemists gave them a sense of empowerment and control.

Issues Revisited

Multiple Representations

A glance at any general chemistry textbook shows a multiplicity of representations used in different contexts with no correlation between representations or why any one was chosen. Some students may switch contexts easily; most do not integrate the knowledge or extract from them representations to ideas experts derive from them. Experts do use different models and switch easily between them when solving problems. Without the ability to cross-reference, novices tend to believe each representations as a complete model of the concept being illustrated (Reif, 1987). Redundant knowledge allows the expert to check the accuracy of each line of reasoning (NSF 92-4, 1992).

Even if our interest is not to form experts, understanding the process may help us think through issues of constructing knowledge. The process remains, in fact, one of apprenticeship and practice. One may even extend the analysis and separate the education of a scientist into education in techniques—technos: craft, artifice—where formal methods (courses, books) dominate, and one of application of the techniques or education in methods—meta: beyond, hodos: way. In the last one, practice and apprenticeship still play a fundamental role.

We need to develop better techniques for sharing multiple mental models between experts and students, and to help students develop the ability to choose a proper model. It is a truism that the way we teach is conditioned by the tools that students have to solve problems. Computer graphic representations are being used to show students different ways of seeing diagrams (Brasell, 1987). Yet students develop intuition by doing, not by seeing, and this is precisely what the new technologies allow us to consider (Smith & Jones, 1989). The ability to understand how laws and principles (of science) behave in a new domain, not only what the laws are (intuition), is an important goal of modern science education (diSessa, 1987). This ability is at the core of the significant changes brought by HPCC to science, particularly in discipline areas where microscopic phenomena control macroscopic, observable behavior.

Role of Tools in Science

An analysis of the use of tools in science is useful in understanding how to define an enabling language to be shared by experts and novices. To understand tools it is useful to categorize them according to their relation with content: tools with specific content vs. tools without specific content.

Tools that incorporate content-specific information are usually discipline-oriented, require prior discipline-specific knowledge, and are useful for categorizing observations and analyzing information (e.g., chemical notation language and chemical algebra). These tools differ from discipline to discipline and do not encourage integration of process understanding. Tools without any content-specific information are interdisciplinary and are useful for understanding and integrating observations (e.g., language and mathematics). In education, computers are often used as tools with content. In science, a conceptually

significant use of computer visualization is an interdisciplinary aid to understanding and communicating (as a tool without content).

It is our contention that general, interdisciplinary, content-free tools, used and controlled interactively by the learner in the context of guided exploration of interpretations and models, provide a mechanism for constructing knowledge and for sharing methods and representations with the experts that developed these interpretations. This is in contrast with education software or courseware which may help in a different context. "Tools" enable the transference of concepts and modes of thinking between disciplines, and may provide a richer basis for constructing scientific knowledge.

Visualization of Simulations vs. Animations

Linn and Eylon (1988) refer to the study of electrical circuits where geometrical orientations and relations of the graphic representations of the circuits often took precedence over the technical, underlying principles—e.g., in-parallel or in-series resistors. This points to an analysis of the different meanings associated with the word visualization.

Visualization, whether it refers to output (data) or processes, has been used to indicate either animation (the arbitrary representation of an explanation) or process- or data-driven interpretations; this last meaning is the one we use. For example, the process of iterative inversion of a matrix of random numbers may be visualized by assigning colors to numbers and seeing the evolution of patterns into a diagonal. For clarity, in this presentation we use "visualization" as a short hand for "visual interpretative experimentation." In both animation and visualization the student may have control of the process, though often in the case of animations he or she does not.

Visualization has two aspects which are now converging. The most visible visualizations have an aspect of animation—assigning discrete surfaces to patterns in data—while software is now becoming available to help the individual scientist or student develop visualizations for exploration—development of a visual language. There is a great danger that the beauty and "neatness" of one-time-only passive visualizations and animations will lead novices to see truth where there is none; the process of visualizing data is as important as the result. The use of low-level, desktop-based, manipulation of numbers as color maps for exploration and understanding has the greatest potential education.

Simplifications vs. Complexity of Real Phenomena

Some concepts are more complex when simplified. A classical example is the interpretation of Vitamin B12 crystal structure by Dorothy Hodgkin. Her group etched slices of X-Ray spectra on thin glass slabs and collected the slabs into a three-dimensional solid. Only then were they able to account for the effect that heavy atoms on one slice had on the pattern etched in other slices. The structure was unsolvable in the two dimensional, simplified approach, yet manageable in three.

We often present students with similar "simplifications." Some, by virtue of our reliance on the printed page (or the projected screen). Others, by virtue of our use of model simplifications that seem real to the students while being only a shorthand notation to be used or discarded as needed. To understand science outside the classroom, students need to understand how complexity (often, nonlinearity) affects the models used to explain concepts. They need to understand how simulations are affected by approximations in the calculation and in the model, and understand the difference between simulations (compu-

tational experiments) and laboratory experiments. Normally students do not learn what complexity really means; they do not have the chance to see the models they study crumble and break under the weight of non-linear growth. Every student should have the chance to see a simulation break down in his or her face; this is the most effective way in which he or she may learn how problems scale up.

Modes of Learning

Introducing real-life complexity ties into the desire to show students what science really is and what scientists really do. Scientists do not extract and explore relationships, but the results in the real world of objects represented in those relationships. It has often been remarked that the way we teach science (technically as well as sociologically) (Tobias, 1990) tends to select for the same ability (mode of learning) common now in the profession: analytical, mathematically oriented. In the case of chemistry, for example, it is not clear that this ability is more important than the visual ability for three dimensional imaging. In fact, we doubt that many natural product chemists of the past would feel comfortable with the way high school teachers and chemists teach chemistry.

Exploring "how laws behave" first and allowing students to manipulate the laws may make teaching "what the laws are" more relevant and understandable and increase the sense of achievement of many students. The question then arises, what should we teach first? Will our strategy work better with students that are now (or culturally) excluded from personal achievement in science?

We would like to end with a note sent to us earlier this year by Mr. Rowe:

> I posted color pictures of the images my students generated on the inside of the teachers' lounge door, with a note that the AP Chemistry students had generated these images. A French teacher told me that she was in there after school one day and 2 sophomore students knocked on the door and asked to come in to collect the cans for recycling. She said OK, and went in to the lavatory. When she returned they had the cans in their boxes, but were stopped and staring at the images. They were discussing what they meant—pointing out what must be covalent bonds, and commenting how they must be orbitals overlapping! These are kids that are in General Chemistry, not AP! I guess that is a pretty strong argument that what we are doing is a good idea!!
>
> Now on to solutions!! [One of the future modules will deal with solvation]
>
> Barry

References

Almlof, J., Faegri, K. Jr., & Korsell, K. (1982). DISCO. *J. Comput.Chem., 3*, 385.

Blumberg, F., Epstein, M., MacDonald, W., & Mullis, I. (1986). A pilot study of higher-order thinking skills assessment techniques in science and mathematics. *Final Report, Part 1. NAEP.* Princeton.

Brasell, H. (1987). The effect of real-time graphics on learning graphic representations of distance and velocity. *Journal of Research in Science Teaching, 24*, 385.

diSessa, A. (1987). *Journal of Research in Science Teaching, 24*, 351.

Eylon, B., & Linn, M.C. (1988). *Review of Educational Research, 58*, 251.

Fulfilling the promise: Biology education in the nation's schools. (1990). Washington, DC: National Research Council.

Hesse, J., & Anderson, C. (1988). *Student's conceptions of chemical change.* Paper presented at the American Educational Research Association Meeting, New Orleans.

Hodgkin, D. Seminar Presentation, University of Chicago.

Kozma, R., Russell, J., Johnston, J.,& Dersheimer, C. (1990). *College student's conceptions and misconceptions of chemical equilibrium.* Paper presented at the American Educational Research Association Meeting, Boston.

Larkin, J.H., McDermott, J., Simon, D.P., & Simon, H.A. (1980). Expert and novice performance in solving physics problems. *Science, 208*, 1335.

The liberal art of science: Agenda for action. (1990). Washington, DC: American Association for the Advancement of Science.

Linn, M.C. (1989). Science education and the challenge of technology. *Information Technologies and Science Education (AETS) 1988 Yearbook*, 119.

McCloskey, M. (1983). Naive theories of motion. In D. Gentner & A.L. Stevens (Eds.), *Mental models.* Erbaum Associates.

Mulliken, R.S. (1966). Spectroscopy, molecular orbitals and chemical bonding. Nobel Lecture for Chemistry.

Prybil, J.R., & Bodner, G.M. (1987). Spatial ability and its role in organic chemistry: A study of four organic courses. *Journal of Research in Science Teaching, 24*, 229.

Reif, F. (1987). Instructional design, cognition and technology: Application to the teaching of scientific concepts. *Journal of Research in Science Teaching, 24*, 309.

Report on the National Science Foundation disciplinary workshops on undergraduate education. (1989). Washington, DC: NSF.

Rowe, B. (1992). Orbital insights. *NCSA RealTime, 4*, 4.

Sabelli, N. (1991). Computational science and education: Workshop on the role of HPCC centers in education. NSF Grant ASC-9018011.

Saebö, S., & Almlof, J. (1987). DISCO. *Chem. Phys.Lett., 154*, 521.

Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Jensen, J.H., Koseki, S., Gordon, M.S., Nguyen, K.A., Windus, T.L., & Elbert, S.T. (1990). GAMESS. *QCPE Bulletin, 10*, 52.

Smith, S.G., & Jones, L.L. (1989). The FIPSE lectures: Chemistry plus technology plus teachers. *Journal of Chemical Education, 66*, 3, 8.

To strengthen American cognitive science for the twenty-first century. (1992). Report NSF 924. Washington, DC: NSF.

Tobias, S. (1990). *They're not dumb, they're different.* Stalking the Second Tier Research Corporation. Tucson, AZ.

Appendix
Background: NCSA's role

This chapter is a report of work in progress, an advocacy statement, and what we hope will be part of a continuing exploration by us and others. The work described has been guided by the role that a national center such as NCSA may and should play in education and by the lessons learned from the success, in both education and research, of NCSA's scientific visualizations and workstation-based software tools. NCSA is not a deliverer of education, nor does it have education research expertise; rather, it is most productive as a resource knowledgeable in technology, science methodology, and their transfer.

The National Center for Supercomputing Applications (NCSA), with a staff of approx. 200, is housed in 70,000 square feet of space spread across five buildings at the University of Illinois, Urbana-Champaign (UIUC). NCSA is one of over 20 research programs located in the Beckman Institute for Advanced Science and Technology at UIUC, a broadly-based interdisciplinary research institute.

NCSA was created in 1985 to meet an urgent national goal: to provide a broad base of researchers in American universities with access to supercomputers over a national network to create a large human resource pool in advanced computational science and engineering (CS&E). Focus for the 1990s has shifted to development in new technologies and computing architectures, and to serving the needs of emerging user communities. Current technological development will lead to a scalable computational structure that is software compatible, from desktop workstations to powerful supercomputers.

NCSA is now preparing for the changing world of the next five years: transforming from a "vector processor supercomputer access center" into a "high-performance computing center." The focus will be on exploring new architectures, application software development driven by CS&E Grand Challenges, and human resources development including initiatives in K-12 and undergraduate education.

The work of NCSA in the area of K-12 education focuses on using High-Performance Computing and Communications (HPCC) methodologies to rethink science education and define the role of computational science in education, and on enlarging the constituency for advanced computing technology in education. In both cases, the objective is to use NCSA's expertise to develop prototype methods and materials for dissemination to the wider community. NCSA is in the best position to bring the leading edge of technology to schools since

- NCSA is a common meeting ground for schools and organizations, locally and nationally;
- NCSA has the visibility and resources to make educational technology programs national and widely used;
- NCSA not only uses HPCC technology, it controls HPCC technology for the benefit of education and research.

NCSA is a supercomputing center, and its expertise is associated with supercomputing. NCSA does not define its role or its mission by the equipment in use, but by the methodology—computational science—that the technology has enabled. It is how technology has empowered scientists to "look" at their work that must be understood and transferred. The rate of technology advance is such that our answers may be needed before

we have time to fully understand them; limiting our thought to the workstations available now limits the options open to students in many ways.

In particular, few students have now direct access to workstations of enough power or with the right software and support to engage in significant explorations. High performance computers with professionally supported software are accessible to increasing numbers of students, and such resources may provide an important "leveling of the playing field." The National Science Foundation and the Department of Energy high-performance computer centers support student access, even in the absence of full connections to the national network (Internet). The combination of networking technology and the existence of these centers enable educators to use resources other than those available locally, and thus, expand the options open to all students.

Many groups have pointed out the need to increase the involvement of practicing scientists and engineers with teachers, students, and with the science education process in general. Increasing the familiarity of teachers and students with the tools of expert mentors will serve this need well: "...since science students are not experts, the tools of experts did not automatically impart the problem solving skills of experts. Rather, these tools provided an opportunity for teachers, researchers and developers to focus on roles and materials that would help students to develop complex reasoning skills" (Linn, 1989).

NCSA would like to help understand and implement the complementary roles that affordable computer simulations, modeling, visualizations, and networking (i.e., HPCC technology) may play in bringing the studying of science and the doing of science closer together (American Association for the Advancement of Science, 1990; National Research Council, 1990; National Science Foundation 1989; Linn, 1989). Exploration of science education pedagogy and associated cognitive issues will benefit from integration of educational tools with software tools that experts use. Providing these integrated tools is one of our aims. Defining the capabilities of the tools is a research topic that calls for collaboration between content science researchers, education technologists, and cognitive and education researchers.

Chapter 12

Edu ·ation and Collaboration in an Evolving Digital Cult·re

DONNA J. COX

Scientific visualization is the process of using computer graphics to represent data initially expressed in numbers. This process is not new. It has been emerging since the beginning of computer graphics in the early 60s. Within the last six years, scientific visualization has grown exponentially due to growth in other technologies such as supercomputers, data gathering, and other related fields. The process involves organizing and mapping numerical data into meaningful visual and aural re-presentations. This process has enabled some scientists to refine and comprehend their models. Resulting scientific visualizations can also provide opportunities to educate large audiences, students, and teachers about areas of science and mathematics that have been difficult to conceptualize (Cox, 1987a, 1988a, 1990a, 1991b, 1991c, 1992a, 1992b).

Modelling reality is both an importan· concept in the history of culture (Cox, 1988b, 1989) and a quest in modern science that employs supercomputers and scientific visualization. Today, computational scientists compute quantitative models of physical reality (Smarr, 1985, 1987; Brown, 1987). Scientific visualization provides a means of literally peering into these models by using the computer.

Unfortunately, few people have both the scientific background and the artistic training needed to do this work well. Most artists are not prepared for the technological and mathematical demands of using computer graphics and mapping numerical data that is required for the visualization process. Yet most artists have demonstrated expertise in visual communication and can make significant contributions to the field of scientific visualization in terms of representational experiments. The other side of this educational coin is that scientists are not prepared for the complexities involved in image-making, post-production, cinematography, and animation. The team approach is a possible solution to these research and educational problems. The artist can contribute to the direction of the scientific visualization representation process while working with technically proficient

scientists and computer experts. The Renaissance Team is a group of specialists who synergistically collaborate to expand the range of options available in the quest for solutions to specific communication problems (Cox, 1988b). The term "artist" is used here in a broad sense. The artist might be a cinematographer, a graphic designer, an industrial designer, or a person wearing a different label that has demonstrated talent for visual communication. When the "art" of visual-thinking is applied to scientific visualization, its function has been labeled information design. The complementary abilities and perspectives among artists and scientists can enhance one another, and this interaction is the raison d'etre of the Renaissance Team. There appears to be a movement away from the team approach toward the creation of tools that allow scientists to be self-sufficient in scientific visualization.

I will address some important issues in scientific visualization and computer graphics education. In the following, I explore areas where scientists and artists can continue to collaborate in scientific visualization and educate, raise awareness, and transfer technology.

The Renaissance of Visualization

Renaissance artists represented and documented their world's every visual element, giving particular attention to detail. Through this imitation of life, they believed that they might capture the essence of life and reveal the invisible laws of nature. Such philosophy set the stage for the Scientific Revolution and the principles of modern scientific methodology that include direct, objective observation of reality and the recording of factual data to form scientific hypotheses. Renaissance artists fostered this approach and worked with biologists and physicians to create a plethora of interdisciplinary books on anatomy and botony (Ronan, 1982).

The classical ideal of striving to imitate visual reality is found in the history of computer graphics and science. Computer graphics researchers have developed algorithms and images that represent what the human eye optically perceives. Computer-synthesized photorealistic imagery requires high-performance computing and advanced algorithms. These graphics algorithms often include simple laws of physics and procedural techniques to create the illusion of realism. In fact, one might say that many computer graphics images provide a "visual proof" of relatively simple equations that mimic light, surfaces, reflections, etc. These visualizations that mimic photorealistic imagery are often referred to as "visual simulations."

An important distinction must be made here between "making things look real" and scientifically computing "models of reality." The primary goal of Computational Science is not merely to simulate the optical appearance of reality, but to simulate models that precisely and completely describe real physical systems. Computational Science is directly linked to the supercomputer revolution because of the necessity for high performance computing environments that provide maximum computer memory, speed, disk store, and networking bandwidth to model the complexity of dynamical systems. These computational simulations produce large data sets, and scientific visualization provides a way to visually explore the simulation data.

Computational science and supercomputing are not the only reasons that the field of scientific visualization is growing very rapidly. Another driving force is the terabytes of data

being collected today with many observational stations on earth and in space. Human beings are collecting data at a rate beyond what we can actually study. Supercomputers have become necessary for computational science and scientific simulations, as well as processing data to create visualizations.

The task of modelling a complex dynamical system such as the biosphere and visualizing the results requires more computing power than has currently been invented. It is the hope of most computational scientists that supercomputers will get bigger and faster to accommodate the modelling of reality. This "reality" is epistemologically related to the Renaissance idea that the essence of life can be captured in the imitation of life, for in computational science the accuracy of the model, its nearness to replicating "real life," denotes its degree of "realism." One of the oxymorons that has evolved from these ideas is a "real simulation."

The current popularity of scientific visualization is the relatively recent recognition by the scientific research community that visual apprehension is far more acute than the ability to assimilate the same information expressed only as numerical data. The physical and conscious visual processing mechanisms enable humans to actively create and give meaning to the world about them (Friedhoff & Benson, 1989). Historically, many advances in technology are coupled to advances in imaging techniques due to this human predisposition to visually probe, capture, and make sense of the universe. Compare the crown of a splashing milkdrop, frozen by ultra-high-speed photography, to the infinitesimal corona of a cosmic ray impact on a copper plate, visible through the electron microscope, to the perimeter of a massive crater left by a meteorite on Mars, revealed through a high-power telescope. These cosmic splashes are far removed from one another in time and space, yet humans can immediately see a connection. An observer can extrapolate from these images to understand the concept of a "splash." To intellectually extrapolate among these "cosmic" splashes is high-order visual thinking.

Visual thinking has been demonstrated throughout the history of art and education. The capacity to visually extrapolate between observable data and computed data is crucial to science. Scientists compare familiar observable phenomena to numerical simulations that describe the physical laws.

There has evolved a deep chasm between the education methodology art versus science. Artists are not prepared mathematically or technically for todays high-tech digital culture. Likewise, most science students are not provided with the visual thinking skills that are often demonstrated as necessary in developing scientific visualizations. The educational system has ferreted out the visually-oriented people from the technically-oriented people. Computer graphics is a field where these individuals and their skills converge. The computer graphics technology has provided a neutral ground where artists and scientists share the same tools. As it was in the Renaissance where artists and scientists shared tools and worked in tandem, so today we can find interdisciplinary teams devoted to scientific visualization.

Will Renaissance Teams Continue?

The educational requirements for mathematical and technical skills have historically contributed to the chasm between the arts and the sciences. The requirements are so fundamentally different between these areas in universities that students learn in ways that are mutually exclusive. Art students are not required to have any serious level of math

or computer skills. Science students are required to focus on skills that do not encourage visual thinking. Since the invention of computer graphics, artists and scientists have begun to converge through the use of similar graphics tools. Likewise, the growth of scientific animations have brought together artistic editors, scientists, and computer animators in Renaissance Teams.

Successful teamwork requires a group strategy and the adoption of behaviors by each member of the group that promote effectiveness: first, the team must identify a common goal (e.g., scientific visualization) and should contain the smallest complement of members capable of most efficiently accomplishing that goal; second, each team member must recognize that the project will be personally rewarding, must acknowledge every other team member's contribution to the project, and must exhibit respect for other team members and be willing to learn from them; finally, each team member must be given appropriate credit when results of the project are presented. It is a challenge to organize such groups of people who adopt all of the above behaviors. While the above is a generic formula for success, most people who are required to work on teams rarely exhibit all of the above behaviors.

The author has been successful with Renaissance Teams in the research and development of scientific visualizations. Part of the success has been because of the informal, academic nature of the projects and also because these teams have involved individuals with similar value systems and mutual respect. Such teams often evolve into friendships. Collaborations among team members have generated a range of visualization productions and tools that span many disciplines. The choice of how the data gets mapped to what type of visual representation is extraordinarily complex and open-ended. The visual representation choice or the interface design can influence how one understands the data. These issues can call to question ethics, truth, and beauty in scientific visualizations (Brown, 1989; Cox, 1987b, 1990b, 1991a, 1991d, 1994; Ellson & Cox, 1988; Onstad et al., 1990).

As scientific visualization has proliferated, many scientists have requested, and justly so, to be self-sufficient in the visualization process. A scientist once said to me that his goal was to embed my talents into the visualization software so that the Renaissance Team was not necessary. This embedding of talent is resulting in useful visualization software which might possibly be the demise of Renaissance Teams as we know them today. I have generated a taxonomy of visualization techniques and a classification of research that are provided at many Centers. The following is an edited excerpt from my contribution to the National Center for Supercomputing Applications (NCSA) Program Plan FY93:

> The NSF Supercomputer Centers have been international leaders in scientific visualization. A primary reason for this accomplishment is because the Centers have provided a broad continuum of resources from low-end desktop tools to modules for data flow environments to remote-user facilities to high-end production visualizations. The value to the scientific community and transfer of technology as a whole can, in part, be measured by the 100,000s of people accessing visualization tools via the internet as well as millions of people who have been exposed to science and high performance computing via televised programming and scientific visualizations.
>
> Over the past seven years of scientific visualization escalation, we have discovered a broad class of functionalities, and each class makes an important contribution to the field. These can be categorized as follows.

1. Low-end interactive, desktop graphics/audio tools for data analysis (NCSA Suite, Data scope, Spyglass). These types of tools have often been the fast and easy road to discovery in data. These tools generally:

- are easy to learn;
- provide a very short time loop between human and data;
- are supported on relatively inexpensive, well-established user platforms and thus provide a portability not found in specialized interactive tools;
- provide functionality for multiple disciplines;
- provide both sonification as well as visual data exploration;
- have visual representations and/or functionalities often adapted from more expensive platforms/productions;
- provide flexibility for human communications not currently found in specialized interactive tools;
- lend themselves to education;
- do not provide extremely refined imagery ("extremely refined" here is used as a term to describe imagery that is smoothly rendered with high production values and requires batch mode animation capability. This type of production imagery cannot currently be computed in real-time).

2. Specialized interactive tools for visualization and data analysis (RIVERS, SDSC Volumetric Visualization, Virtual Reality, High-Definition workstations). Generally these tools:

- have addressed specific user needs in the original development or design;
- require users to be proficient, mentored, or trained;
- are developed on expensive, highly-specialized platforms;
- address a specific discipline or type of application;
- require systems development or support;
- are expensive to maintain;
- provide interactivity that animations do not provide;
- provide both sonification as well as visual data exploration;
- have features that have migrated to less expensive platforms;
- provide greater speed than low-end tools and increase efficiency of scientist;
- require dedicated software development and maintenance.

3. Data flow environments (e.g., AVS, SGI Explorer, ApE) have been one of the graphics industry's answers to specialized scientific visualization tools. These data flow environments generally:

- provide users with the flexibility to customize the interactive environment;
- provide a mechanism for portability of visualization modules;
- require a level of sophistication in software development in order to develop modules;
- provide a mechanism to interactively explore relatively large data sets;
- are useful for personal discovery process as well as presentation to peers;

BEST COPY AVAILABLE

- require specialized platforms for the modules (there is intense competition for one of these data flow environments to emerge as a standard);
- provide an interactivity that animations do not provide;
- can be customized for multiple disciplines;
- do not provide extremely refined imagery.

4. Peer batch-mode visualizations (not interactive and result in a video tape animation). These animations generally:

- are specialized for scientific peers and are intended for small audiences;
- are developed by the scientist for a specific discipline;
- are algorithmically tied to the simulation and do not explore alternate visual representations of the data;
- provide the scientist with a mechanism to research his/her data that cannot be built into a tool;
- require software support and maintenance that is not transportable across many platforms;
- require the viewer to have intimate knowledge of the subject in order to fully understand the visualization;
- require the scientist to have graphics visualization knowledge to some degree as well as image file formating capability;
- do not require verbal scripting, expert packaging, or detailed refinement of visual quality;
- can feed directly back into visual representations used for interactive tool development;
- can use LVR, Video Mac frame-recording, Macro-mind director scripting, and editing (however, image quality is less than Abekas frame-recording);
- do not have production quality that make them useful for television, movies, or advanced presentation graphics;
- can sometimes be packaged into formats for the following high-end presentation production visualizations;

5. High-end presentation production visualization animations generally:

- are batch-mode using high-end software and hardware;
- require computer graphics and/or communications experts to assist scientists in the production;
- provide flexibility to explore visual representation techniques that data flow environments and interactive tools cannot provide to date;
- allow a relatively sophisticated presentation of data, often used for conferences, competitions, television, and movie presentations;
- are used to develop the most advanced visual representations possible;
- provide visual representations that are sometimes incorporated into interactive tool development or data flow modules;
- are useful to multiple disciplines;
- are educational, informative, and raise awareness for large general audiences;
- are choreographed, voiced-over, scripted, edited, titled, and post-produced;

- involve production of extremely refined imagery and require attention to detail;
- can incorporate audio as well as visual exploration of data;
- require a production environment with software support and maintenance;
- can document and/illustrate important results and concepts in a field;
- are appropriate for high-definition imagery.

The above represents a continuum that enables development of advanced visualization techniques across disciplines and empowers individuals using visualization as an indispensible tool for data exploration.

Many of the above visualization tools/techniques do not require teamwork, though teams are often involved in the development of the software tools. While artists can contribute on many levels, it seems that there are definite trends in scientific visualization: to shorten the feedback loop between scientist and data; to have interactive, real-time tools to explore data; and to move away from batch-mode animations that require extensive rendering time. These trends that might allow the scientist autonomy, also do not encourage interdisciplinary collaboration.

Strategy for Visualization in Teaching

In the Renaissance, collaboration was a working model. However, by the turn of the 20th century, art and science diverged along separate paths (Careri, 1986; Weininger, 1990). Today, most scientific research is performed by individuals separated by disciplines and specialities. Only in "big" science has there remained a need for researchers to collaborate and pool resources on large-scale projects like the space program or supercolliders (Baker, 1986). Yet, collaboration often plays a major role in industrial careers, art forms, filmmaking, computer animation, software tool development, and hardware design. In most higher education, undergraduates and graduates do not get a real-world understanding of the importance of collaboration or how collaboration will play a role in career development. In particular, collaboration between the arts and sciences within a university setting is almost nil. The process of collaboration is a skill that can be coupled to computer graphics and scientific visualization education. In my university courses, I put teams together of computer scientist/engineer students and art/design students to develop specific projects in scientific visualization and scientific communication (Cox, 1991c, 1992b). My courses are intensively rigorous, to the point that the Computer Science department gives undergraduate and graduate credit. The production animations/visualizations from this course are educational and entertaining, and they have won international awards.

I contend that scientific visualization is not a product; rather, it is a process. Students and researchers need to be taken through this process with the understanding that they will grow out of old techniques, into new ones. The following is a reorganization of the former taxonomy. We can see that existing tools and animations are valuable for a range of educational experiences.

1. Interactive, desktop graphics/audio tools for data analysis (NCSA Suite, Data scope, Spyglass) are generally extremely useful for education because they are relatively easy to learn and provide a very short time loop between students and data. They are generally supported on relatively inexpensive, well-established user platforms such as MacIntoshes which many schools have. Students tend to retain information and interactively learn faster because of direct "hands-on" experience of using the computer for exploration. The primary disadvantages are that this level of computers can be slow, and they definitely need teacher preparation for classroom instruction. This requires teachers to learn the software, in addition to having knowledge of the subject.

2. Specialized interactive visualization tools for data analysis (RIVERS, SDSC Volumetric Visualization, Virtual Reality, High-Definition workstations) are generally not useful for the classroom because they are too specialized and require expensive platforms. However, if one documents these environments by using them as demonstrations for educational video programs, the specialization allows one to educate students about potential future areas of research.

3. Data flow environments (e.g., AVS, SGI Explorer, ApE) unfortunately require advanced graphics workstations that most schools cannot afford. However, they would be generally useful for university education because the modules could be written by advanced students or scientists for the educational setting. They provide a mechanism to interactively explore relatively large data sets and would accelerate the learning process. Thus, they would provide an interactivity that animations do not provide and they can be customized for multiple disciplines.

4. Scientific peer batch-mode visualizations that result in a video tape can be useful for education; however, they are generally not as refined as the above production visualizations. They are generally targeted for a specific scientific peer audience and require some type of explanation from either a teacher and scientist to make them useful for student education. Unfortunately, unless they are packaged with educational materials or explanations, they would require the teacher to understand the subject fully in order to teach with the animation. If they were repackaged, they could be transferred to other multimedia formats as the above production visualizations. Most of these types of animations are only understandable by peer scientists but could be used as raw material for an educational setting or perhaps packaged into presentation production visualizations.

5. Presentation production visualizations require animation recording equipment and production capabilities that schools cannot usually afford. However, those schools that have animation capabilities would find these productions valuable as "hands-on experiences" for students. These types of experiences provide flexibility for students to explore visual representation techniques.

   If schools cannot afford animation production equipment or computers, they can often use video tape players to show students animation productions that already exist or are produced elsewhere. These animation video tapes can be useful as educational tapes in the classroom. These types of productions which demonstrate and explain science and concepts allow a relatively sophisticated presentation of data. This is why these

animations are often used for conferences, competitions, television, and movies. They are useful to multiple disciplines and require interdisciplinary teams to produce the high-quality visualizations. They involve voice-over scripts and are presented in an educational and informative way for the large, general audience. They can document and/illustrate important results and concepts that would be impossible or difficult for other types of interactive tools. They are packaged material that facilitates the teacher in a classroom.

Interdisciplinary collaborations have by far produced the most exciting and stimulating animations. They have become intellectually rich visual communications. Presentation production visualizations have resulted in outstanding works that will go down in the history of art/science endeavors. These productions are akin to great collaborative films (see Color Print 9). It is here that the animations are extremely valuable for education and informal science. Truly these types of productions can contribute to the "storytelling" of science and mathematics. This type of "storytelling" is entertaining and inspires youth to participate and learn science and mathematics. Scientific discoveries that are only published in rigorous juried journals remain esoteric to the masses. Only a small percentage of people per year will ever read these types of peer-reviewed articles. Presentation productions help make abstract science and math much more accessible to larger numbers of people.

The continuum is complete when the educational, "hands-on" process is provided in the classroom. Low-end, interactive computer tools provide an excellent way to teach mathematics and science because of the interactivity and the participatory nature of the tools. Both interactive tools and production visualizations provide a variety of ways to transfer technology to students. One can reach many students through television or video programs, but for retention and active learning, students must have "hands-on" access to computer graphics tools.

Collaboration and Networks

In collaboration with Robert Patterson, we have produced several data visualizations of the NSFnet internet growth and traffic. Robert developed software, co-directed, and animated the visualizations. We have designed animations for both general and peer audiences. The data for these visualizations was accessed over "The Net" via file transfer protocal (ftp) with the help of Merit Network Inc. The High Performance Computing and Communications initiative has placed a great importance on network and communications development that will link educators, researchers, and industry. Robert and I have visualized some of the first animations of the internet that demonstrate rapid growth and very heavy, sustained activity.

The connection space (see Color Print 10) represents virtual connections among sites to a T1 NSF backbone. The sites might connect to this backbone via ethernet, fiber, cable, or other physical connections with variable bandwidth speeds. Once on the T1 NSF backbone, information flows at a relatively very fast rate. This backbone will soon be upgraded to T3 and the speed of information flow will increase. These connections represent a space where people using computers can communicate, send/receive data, and control computer programs remotely. A mass communications subculture has evolved from this

technological capability that would allow students of all ages to access information databases and digital images from any geographic location. Scientific visualization will inevitably become a part of research, education, and the information exchange futures. Remote collaboration over "The Net" will become easier and more productive within this evolving digital culture.

Conclusion

Collaboration across the arts and sciences has waxed and waned over the history of civilization. Scientific visualization provides a new role for the artist/scientist as collaborators in important areas of scientific communication. Images are instruments of this communication. Scientific visualization is not a product, it is a process that allows an enormous amount of information to be understood. Skills possessed by artists extend the storytelling capability of rigorous science and enhance communication of abstract concepts. While these images will remain documents of the collaborations that produce them, the deeper cultural significance of, and the subtle social effects of artists working with scientists, may never be completely documented.

Scientific and data visualization will continue to grow; the technology on every front is pointing in that direction. Visual communication of the sciences lends itself to interdisciplinary collaboration and education. As educators, we must not only encourage technical and scientific skills, we must also encourage skills to enhance the collaborative and communication process. In the future, collaboration among individuals may take a new form as a result of the evolving communications technology. Networking and visualization will be integral to the information age where collaboration over "The Net" is part and parcel of our new digital culture.

References

Baker, W. (1986). *NASA: America in space.* New York: Michael Friedman Publishing Group.

Brown, M.D., DeFanti, T.A., & McCormick, B. (1987, November). Visualization in scientific computing. *Computer Graphics, 21* (6).

Brown, J.R., & Cunningham, S. (1989). *Programming the UserInterface: Principles and examples.* New York: Wiley.

Careri, G. (1986). Art and science in search of non-visible worlds. *Leonardo, 19*(4), 275.

Cox, D.J. (1987a). Computer art/design curricula in universities: Beyond the traditional approach. *Teaching computer graphics: An interdisciplinary approach* (pp. 207-223). Siggraph '87 Educator's Workshop Course Notes. Published by the Association for Computing Machinery's Special Interest Group on Computer Graphics in cooperation with the IEEE Technical Committee on Computer Graphics.

Cox, D.J. (1987b). Interactive Computer-Assisted RGB Editor (ICARE). *Proceedings for the 7th Symposium on Small Computers in the Arts* (pp.40-45).

Cox, D.J. (1988a). Renaissance teams and scientific visualization: A convergence of art and science. *Collaboration in Computer Graphics Education, SIGGRAPH '88 Educator's Workshop Proceedings* (pp. 81-104).

Cox, D.J.(1988b). Using the supercomputer to visualize Higher Dimensions: An artist's contribution to scientific visualization. *Leonardo: 21 Journal of Art, Science and Technology*, pp. 233-242.

Cox, D.J. (1989). The Tao of Postmodernism: Computer art, scientific visualization, and other paradoxes. *ACM SIGGRAPH '89 Art Show Catalogue, Computer Art in Context, Leonardo Supplemental Issue* (pp.7-12).

Cox, D.J. (1990a, Winter). Scientific visualization: Collaborating to predict the future. *EDUCOM Review*, pp. 38-42.

Cox, D.J. (1990b) Scientific visualization: Mapping information. *Proceedings, AUSGRAPH '90, 1990* (pp. 101-106). Published by the Australian Computer Graphics Association.

Cox, D.J. (1991a, November). Beyond visualization: Mapping information. *Supercomputing 91 Tutorial Notes.*

Cox, D.J.(1991b, June 6). Collaborations in art/science: Renaissance teams. *The Journal of Biocommunications, 18*(2).

Cox, D.J. (1991c, July). Interdisciplinary collaboration case study in computer graphics education: "Venus & Milo." *ACM SIGGRAPH Computer Graphics, 25* (3), 185-190.

Cox, D.J. (1991d). Scientific visualization: Supercomputing & Renaissance teams. *Proceedings of the 12th New Zealand Computer Conference* (pp. 157-171).

Cox, D.J. (1992a). Caricature, readymades, and metamorphosis: Visual mathematics in the context of art. *Leonardo, 25* (3/4), 295-302.

Cox, D.J. (1992b). Collaborative computer graphics education. In S. Cunningham & R.J. Hubbold (Eds.), *Interactive learning through Visualization: The impact of computer graphics in education* (pp. 189-200). Berlin: Springer-Verlag.

Cox, D.J. (1994). Glitzy or grungy graphics: Choose your audience. *AAAS '94 Program and Abstracts* (pp. 93-94). 160 National Meeting of the American Association for the Advancement of Science, for panel "Science, Lies, & Videotapes," San Francisco, CA.

Ellson, R., & Cox, D.J. (1988). Visualization of injection molding. *Simulation: Journal of the Society for Computer Simulation, 51*(5), 184-188.

Freidhoff, R.M., & Benson, W. (1989). *Visualization: The second computer revolution.* New York: Harry N. Abrams.

Onstad, D.W., Maddox, J.V., Cox, D.J., & Kornkven, E.A. (1990, January). Spatial and temporal dynamics of animals and the host-density threshold in epizootiology. *Journal of Invertebrate Pathology.*

Ronan, C. A. (1982). *Science, its history and development among the world's cultures.* New York: Hamlyn Publishing Group Ltd.

Smarr, L. (1985). An approach to complexity: Numerical computations. *Science, 228*(4698), 403-8.

Smarr, L.(1987). The computational science revolution: Technology, methodology, and sociology. In R. B. Wilhelmson (Ed.), *High-speed computing: Scientific applications and algorithm design.* University of Illinois Press.

Weininger, S. (1990, January 8). Science and "the humanities" are wedded, not divorced. *The Scientist*, pp. 15-17.

Chapter 13

The Hypergraphics Honors Seminar at Illinois

GEORGE K. FRANCIS

One edition of Math 198, the Freshman Honors Seminar in Mathematics at the University of Illinois at Urbana-Champaign, is an intensive introduction to real time interactive geometrical programming. Its name, "Hypergraphics," connects to David Brisson's (1978) proposed synthesis of art and mathematics for the purpose of revealing the mysteries of space beyond the confines of our 3-dimensional perception (Banchoff, 1990).

The course is designed for novices, but experienced programmers are welcome, provided they contract for an individual study project commensurate with their skills. Most beginners also reach a level of competence by the middle of the course to complete a project of their own. Students work on Apple IIgs and Silicon Graphics Iris computers. They program in BASIC, Forth, and C. Of course, these languages are augmented by graphics packages. For the first, &-GRAFIX is a machine language extension of Applesoft BASIC which was written by students in the UIMATH.APPLE Lab over the past four years (Sandvig, 1990). The ISYS Forth compiler is the product of a local software engineer (Illyes, 1988). It was developed, to a large extent, with the needs of the Apple Lab in mind. The graphics library on the Iris, known as gl, is such an effective resource that it is possible to learn enough basic C to write a respectable real-time interactive computer animation project during just one semester.

The students in this elective course are generally members of the University of Illinois Campus Honors Program, which selects 500 bright students from a population of 27,000 undergraduates. Thus, the members of my classroom compare well with the students taking similar courses at private universities. For example, a somewhat similar course at Princeton was taught and reported on by Conway, Doyle, and Thurston (1991).

In this chapter I shall describe my course in some technical detail so that the reader may not only profit from my experience but may weigh the basis of my opinions regarding high-

tech education. Respecting the principle that a mathematics paper should always contain "something old, something new, something borrowed, and something true," I include the complete, annotated, 250-line source-code for illiSnail, a real-time interactive computer animation (RTICA) which my students use, study, and modify on the Iris 4D/25TG computers in the Renaissance Experimental Laboratory (REL) of the National Center for Supercomputing Applications (NCSA) of the University of Illinois at Urbana-Champaign (UIUC).

Portrait of an Honors Student

Let me begin, by way of an anecdotal documentation, by sketching the activities of a recent, not atypical student. Pablo was technically a freshman but came to college with an excellent preparation. Together with his prodigious talent, this allowed him to compete with the juniors and seniors for first place in the class of fifteen. Pablo's first "essay" was a whimsical animation in &-GRAFIX of swimming fish blowing bubbles. His second was the best of only three solutions for an assignment to write a concise, recursive program in Forth that draws a Sierpinski Triangle. The class studies a series of very simple programs which are small enough to fit into one's mental "hip-pocket" and can be played on every computer. Pablo's "commentary" on the one for the lesson on Logistic Chaos was to modify it and so draw the well known bifurcation diagram of this famous dynamical system.

Two years ago, my teaching assistant, Glenn Chappell, had brought with him a superb piece of pedagogical software. His program is written in BASIC and 65816 machine language. It is, in fact, an interpreter for a tiny language, CSL, which Glenn invented for simulating the cellular automata popularized by Kee Dewdney in the pages of the Scientific American. Pablo completed the assigned experiments with CSL, comparing them intelligently to their older Forth versions. In my graduate Geometrical Graphics course on the Irises, we develop RTICAs which have a feature for recording the user's activity in a script which can then be played back automatically. Although such student projects are often user-hostile, they deal with interesting topics that appeal to the honors students. The "movie-making" feature makes it possible to incorporate them into lessons for Math 198. Pablo's cohort used the Snailhunt RTICA to explore a certain Möbius band located on the 3-dimensional hypersphere in 4-space (Francis, 1990). (This RTICA is discussed in detail in a later section of this chapter.) Pablo's movie and brief documentation showed an unusual level of curiosity and good mathematical free association. He manipulated the program to produce fanciful shapes reminiscent of public art on the Daley Plaza in Chicago. Working out the answers to the conjectures he made could have become his semester project. But he chose a much more ambitious one: to implement Carter Bay's 3-dimensional version of Conway's Game of Life as described in Dewdney's (1988) popular book, *The Armchair Universe*.

Pablo's project was the most notable achievement in that class. He wrote his own RTICA of a 3-dimensional Life automaton. With only a high school AP-Pascal course behind him, he learned and mastered C/Unix/gl on the Iris and wrote a program few of the students in my graduate course could write. The more ambitious student projects often receive extensive help from the staff. Pablo managed all that pretty much on his own.

Pedagogical Notes

Although Math 198 is similar to Thurston's Math 199 at Princeton (Conway, Doyle, & Thurston, 1991), there are enough differences to warrant a closer comparison of these two courses. In fact, my Math 198 more closely resembles the two-week intensive course, "Geometry and the Imagination," taught by John Conway, Peter Doyle, Jane Gilman, and William Thurston at the Geometry Center, summer 1991. Their course was followed by a ten-week research and training program for some of the high school and college students who took the short course. This permitted the completion of substantial projects, some of which included computer visualization (Marden, 1991).

Math 198 at Illinois is a 3 credit course, though all, including the instructor and the teaching assistant, spend far more time on this course than is customary for an undergraduate course with 3 assigned contact hours per week. Quality education at a large, state supported university requires such dedication and extra effort.

The Project

Each student has a project to complete. The project presentation has an oral component. This takes place during the final week of the semester, seminar-style and with cake and soft-drink refreshments. Typically, the student explains what the program is about and how to operate it. An abbreviated version of hands-on demonstration earlier in the course is followed (sometimes preceded) by a 10-20 minute lecture at the white-board. The demonstration and segments of the mini-lecture are videotaped. The demo taping is occasionally staged and repeated to improve its quality. However, little of the mini-lecture and none of the discussion is taped, to encourage both presenter and audience to express themselves freely. Our unexpected experience is that, unlike the author's generation, members of today's TV generation show practically no camera shyness or stage fright. Also, they know how the videotapes will be used: They are shown at the Honors House on special occasions—for instance, to visiting parents of prospective participants. Older students, who are helping out during these orientation sessions, are delighted to watch their colleagues "perform" their final for Math 198. Last year's tapes are also shown in class to explain to the students what is expected of them.

The Grade

A second major difference from the Princeton course is how the final grade is determined. A course like Math 198 is unsuitable for either a pass/fail or a standard grading scheme. The temptation to procrastinate or merely audit such a course is too great, especially for the freshman and sophomore. On the other hand, inhomogeneity of preparation, experience, and motivation precludes competitive examinations and comparative evaluation. Instead, each student receives a "contract-grade" according to the following announced and periodically repeated formula. Once the student has completed the basic assignments and tutorials, he has earned a "gentleman's C" for the course. All that needs to be done for an A is to complete the semester project. A student whose project is well started but incomplete receives a B. This can be changed to an A once the missing work is submitted.

I have never had to give the gentleman's C, and the drop rate is about 1 to 2 students per class of 15-20 students. Of course, everyone is individually evaluated to facilitate

writing recommendations, which many students in the Campus Honors Program eventually request. Also, generous praise and encouragement are offered privately as needed.

The plan for the project is negotiated with the instructor. The pre-proposa' proposal, progress report, and (rarely) preliminary draft are carefully monitored and commented on. The principle guiding the choice and ambitiousness of the project is for the student to apply newly acquired mathematical and computing skills. The project is complete insofar as the program works, has been publicly presented, and the written documentation is acceptable. The latter includes a one-page operating instruction, a careful specification of the hardware configuration, a narrative essay with bibliography on the mathematics, a hand-annotated printout of the program, and a technical note on computational difficulties that were solved or remain to be solved by the next student building on the present project. The class materials are to a large extent the work of previous students, often with emphasis on their shortcomings. Thus some of the best projects each year are corrected continuations and extensions of previous projects. The project is treated as a contractual agreement to produce a certain piece of work by a deadline.

Manipulatives

The Princeton course makes excellent use of geometrical artifacts, including mirrors and construction kits. We make no extensive use of physical models or experiments, except on the computers and, to a much lesser extent, with video equipment. This is entirely the consequence of our severely limited physical facilities at Illinois.

The Journal

In the Princeton course the student keeps a bound journal into which assignments, class notes, and other appropriate items are either pasted (if completed on a word-processor) or entered by hand. While I have always encouraged my students to keep a 3-ring notebook for handouts, clean copies of their class notes, homework, and tests, I had assigned the keeping of an "intellectual journal" only last year after learning about its use in the Princeton course. In the fall we assigned journal keeping to some 100 students, who were preparing to become elementary school teachers, in the lecture/lab course on "Experimental Arithmetic" (Francis, 1992) which also has a programming project component. The logistics of this multisection course, the immaturity of the students, and our neglect to collect the journals regularly and monitor their content, led to the failure of this first experiment. On examination at the end of the semester, 90 percent of the journals were nothing more than daily diaries which recorded the trials and tribulations of an unfamiliar and difficult lab course. Except for the psychological benefits of catharsis for the students and scathing criticism for the instructional staff, these journals were a waste of resources. Quite the contrary was the case for the journals kept by the Math 198 students the following spring. I explained from the start the Princeton model of the journal, including its pros and cons, and invited my students to experiment with their own format. The only requirement was that the hard-bound journals (no fair tearing out pages) had ample margins and blank even-pages for comments and corrections. The journals were collected and read two or three times during the semester and commented on, copiously in some cases. A written exchange developed between student and teacher in a few journals, an almost Victorian dialogue of glosses. On the first round, about a third of the students had nearly empty journals or started writing a diary instead. Most of these had mended their way by the time of the

second reading. Some of the journals were astoundingly good right from the start. Of course, one must not forget the exceptionally verbal, high honors cohort taking Math 198.

The journals proved to have unexpected benefits also for the professor, not the least of which was the fact that a 10x15 inch, black, hardbound journal is difficult to lose or misplace. Secondly, it provided an opportunity for individual instruction. Erroneous or incomplete journal entries prompted me to write a mini-lesson right into the journal. This information and elaboration beyond the class instruction went directly to the interested student, without wasting other people's time. Finally, cumulative progress could be monitored over the semester without having to decipher the numerically encoded entries of a gradebook. I even got a second chance to correct my own misleading "corrections" on rereading them at a later time.

Facilities

Math 198 has been taught for three years in succession in an essentially identical fashion. This particular configuration of hardware, software, students, instructors and content was based on the experience with different configurations of related courses taught under the auspices of the UIMATH.APPLE Lab since 1983. It was therefore uniquely suited to its academic environment. Without similar experimentation and fine tuning it is unlikely that such a course can be successfully taught in another environment. Nevertheless, we hope that a careful description of its configuration below will be of use also to someone planning such a course under their own circumstances.

The Student Cohort

Math 198 is for students in the Campus Honors Program, but others with comparable credentials may also take the course. The University of Illinois Campus Honors Program admits, on a competitive basis, circa 500 students from an undergraduate student body of ca 27,000. Or, approximately 100 students from over 600 applicants join the program each fall. (The difference is made up of students who join the program at a later time.) The quality, motivation, and preparation of these students therefore is not dissimilar from those for whom the Princeton course was designed. Math 198 is officially an elective for freshmen. Its curriculum is the instructor's choice. Since the course under the present discussion, and its predecessors, is the only version of Math 198 which treats programming graphics computers, no ambiguity results from referring to it simply as Math 198.

In a class of 15-20 students, a third tend to be freshmen, the others range over all three remaining years. Typically, five are novices with respect to computer programming, four are so proficient that they volunteer to help train the novices, and the remainder can program in at least one language on at least one computer. Occasionally, a freshman is among the computer proficient, but juniors or seniors who are computer novices are discouraged from taking the course. The students are usually science or engineering majors, though there are always one or two from the humanities or the fine arts. There have never been any students from agriculture, commerce, or the social sciences. Two or three students are women.

Not surprisingly, all students have had calculus, trigonometry, and analytic geometry, at least in high school. The majority are concurrently enrolled in a middle level course in

differential equations, linear algebra, geometry, physics, or computer science, so that their projects are almost invariably related to topics studied in these related courses.

Instruction

The professor and his teaching assistant both meet the students for an average of five hours a week in lecture and lab. The course carries 3 credits, but the schedule is so arranged that the lab component is contiguous with the formal instruction. The topics are strongly modularized so that they can be variously selected and arranged to maximize their relevancy to the students' current interest and capacity. A 5-10 minute introduction is followed by a 15-30 minute hands-on tutorial on the computer. This can be in the form of operating, analyzing, modifying, and experimenting with a pocket program (see below) on the Apple IIgs. On the Iris it usually means operating an RTICA to perform a particular set of experiments and recording the outcomes (observations) on a work sheet or in a notebook.

Hardware

The computers used at the high and the low end are standardized. Each student has access to a fully networked Personal Iris 4D/25TG in the Renaissance Experimental Laboratory (REL) of the National Center for Supercomputing Applications (NCSA). These Irises have 24 bit RGB framebuffers with a 24 bit hardware Z-buffer and graphics accelerators. This configuration suffices for real-time interaction with fully animated, rendered and lighted scenes containing 1 to 5 separate graphics objects.

Half the class sessions take place in REL, the other half in the Apple Lab. There the students use 20 independent, 9-year-old Apple IIe computers which have been upgraded to be a IIgs with (effectively) 4-bit RGB color or 4-bit gray- scale Z-buffer.

In addition, many students own popular micro computers, and the University provides access to Macintosh and IBM sites. While these sites are very useful for word processing, they are uniformly useless for Math 198. Students with computers in their rooms are permitted (weaker students are encouraged) to structure their projects in a way that maximizes their using personal equipment for graphics and mathematical algorithms. Of course, it remains their responsibility make sure that the project can be demonstrated to the entire class on some computer in a publicly accessible location.

Software

On the Iris we use a proprietary but close variant of the Unix operating system, the MIPS C-compiler, and the standard graphics library supplied by Silicon Graphics. All students use real-time interactive computer animations (RTICA) written by and for students in the graduate course I teach in REL, or as projects by previous undergraduates in Math 198. (An example of such a program is listed and discussed in the last section of this chapter.)Those with sufficient experience (beginning with Pascal proficiency) are encouraged to master the rudiments of an editor (vi, emacs, or jot) to try their hand at programming. For novices in C, Unix, and gl, there is a graduated collection of programs to study and modify. For the more advanced student (a few of whom already have some experience programming an Iris) there are projects by previous students to study, improve, and frequently rebuild from scratch.

For the Apple IIgs we have a considerable accumulation of student and instructor developed software, most of it in 65816 machine language, which extends the native Applesoft BASIC resident in the IIgs ROM. The most popular of these is Sandvig's gs.amper.new. This contains a large number of graphics primitives which fit into an ordinary Applesoft program much the same way as calls to the Iris graphics library fit into a C-program. Much more versatile is Robert Illyes' (1988) Forth compiler, ISYSFORTH/GS, which is specifically adapted and optimized for Apple IIgs graphics. Both languages were developed with the needs of the Apple Lab in mind, and both permit us to use the Apple IIgs as a simple, understandable "toy-version" of a "grown-up" graphics computer like the Iris.

Applesoft BASIC extended with &-GRAFIX is commonly used for projects by people who have never programmed mathematics before. Forth is mostly used to tease the last ounce of performance out of our Apples, and to introduce certain concepts such as cellular automata, recursion and object oriented programming in an analytical fashion. In Forth, the student can reach every corner of the computer and even mess with the machine-stack, something which it is not recommended they do on Macs, IBMs and Irises.

On their private machines, the students generally program in TURBOPascal and a variety of C-compilers. A popular project for someone who just acquired an brand new 386 or 486 box with a good but user-hostile graphics card is to write their own high level graphics library, sometimes with machine-language routines for certain primitives (lines, fills, rotations, for example). It should be emphasized here that in the instructional part of the course, geometry is assisted by computer graphics. But, in the projects, it is usually the other way around.

Content

We have developed a number of different techniques of introducing various topics into the course. Here we shall report on only two of these, the pocket program for any personal computer, and an RTICA on the Iris graphics workstation.

Pocket Graphics Programs

These are simple programs that do non-simple things. They also contain working examples of useful techniques. A pocket program expresses one idea in as economical a way as possible within a given language. It is a program one carries around in one's mental pocket, to be produced on demand, and implemented on whatever computer is at hand. It is primarily a didactic instrument. The dozen or so pocket program in Math 198 first of all serve to introduce the student to the major themes of the course. Together, they provide the skeleton on which to hang the concept of computer geometry. Finally, they focus the effort of learning a new language or mastering a new graphics card by way of something concrete to translate or implement. Let me illustrate what I mean by describing two of the pocket programs on adjacent topics.

The Sierpinski Gasket

This is a planar Cantor set obtained by recursively removing the central quarter of a triangle. Connecting the midpoints of the sides of a triangle decomposes the area into four congruent triangles similar to the parent triangle. Inflicting this excision to each of the

three daughter triangles, and their offspring in turn, leaves a figure whose fractal dimension is easily demonstrated. Self similarity shows that doubling the side-size of the Gasket triples the measurable content of the figure. If, by analogy to lines, squares, cubes and tesseracts, "dimension" is defined to be that power of 2 the content of a figure must be multiplied by in order for it to equal the content of a similar figure obtained by doubling its linear scale, then

$$1 < d = \log_2(3) < 2$$

Here is a curious way of generating a Sierpinski Gasket due to Michael Barnsley. For the traditional description, see Banchoff (1990, p. 32). Barnsley (1988, p. 179) invented it to illustrate an Iterated Function System. It can be expressed in eight lines of classical BASIC.

```
10 REM SIERPINSKI
20 X=100:Y=50
30 CLS : REM INIT GRAPHICS
40 FOR I= 0 TO 2:READ XF(I),YF(I):NEXT
50 DATA 0,32, 100,0, 100,63
60 J=INT(3*RND(1))
70 X=(X+XF(J))/2:Y=(Y+YF(J))/2
80 PSET (X,Y) : REM PLOT POINT
90 GOTO 60
```

This program moves a point to a new position which is half way from its current position towards a randomly chosen one of three fixed points. This stochastic dynamical system has a fractal attractor.

This program uses only two graphics primitives: initializing graphics on the Radio Shack TRS-80 Model 100 with the clear-screen command on line 30, and plotting the point (X,Y) on line 80. It is appropriate that the vertices of the triangle be given as an array, line 40, because this simplifies how the choice, line 60, is made each time through the loop. Since line 40 can be written as

```
40 READ XF(0), YF(0), XF(1), YF(1), XF(2), YF(2)
```

it is also a gentle introduction to a counted loop. The compact READ/DATA format is convenient here to express the triangle as a geometrical object in terms of its display-list.

Even at this basic level the program above invites experimentation. Replacing the fixed initial position, line 20, by a user INPUT statement, and later by a random choice, underscores the fact that the gasket is a universal attractor. Altering the number of vertices, lines 40-50, and the proportion away from 1/2 in line 70, leads to some remarkable discoveries. For example, last spring Monica Plisch discovered variants of this iterated function system, whose attractors were other well known fractal figures, such as Koch's Snowflake. A small generalization, using a color computer, leads to a surprising variation discovered spontaneously by many students. It makes an instructive difference whether one sets the color to number J on line 75 or on line 85. The former identifies the three sub-gaskets. The latter, recording the color of the previous choice, gives a visual clue of how

iterated function systems work in the first place. Barnsley's maple leaf (1988, p.108) is not too far away at this stage of the tutorial.

On the other hand, on the Iris it is an easy generalization to 3- and 4- dimensional iterated function systems, which provides a nice motivation to a more formal treatment of a semigroup of contracting affine transformations.

Pocket Dynamical Systems

The exact number and identity of pocket programs used in any given instance of Math 198 is not fixed. Only their use and purpose remains the same from year to year. A large collection of concepts, most of them new to the student, are more effectively taught by way of examples than in a systematic syllabus of abstractions. Training in the vocabulary of comparative anatomy is not needed to enjoy a visit to the zoo. One needs an intelligent arrangement of live examples from each of the major zoological classes, together with brief descriptions that do not neglect the homologies between different species.

More than half of the students in Math 198 major in the traditional (hard) sciences. Thus the notion of a dynamical system is one of the themes to be developed thoroughly. With Hirsch and Smale (1974, p.159) we favor this definition: "A dynamical system is a way of describing the passage in time of all points of a given space ...." The Sierpinski Gasket iterated function system is a kind of dynamical system, albeit a stochastic one. A dynamical system moves a point to a new position according to a rule which depends only on the coordinates of the point being moved. In the present case there are three rules to choose from, and the program uses a generator of pseudo-random numbers to choose which of the three rules to follow each time. When there is only one rule, one says that the dynamical system is deterministic. One calls the stream of fractions returned by successive calls to RND(1) "pseudo-random" because they only seem random to us because we are ignorant of the algorithm that produces them. Of course, this algorithm is designed to mimic a true random number generator as well as the programmer can manage.

The Sierpinski Gasket is the attractor of this dynamical system. Informally speaking, this means that it is a set of points towards which each trajectory (or orbit) tends. The succession of positions taken by a point under the influence of a dynamical system is called the orbit or trajectory of its original initial position. The attractor should also be an invariant set, that is, the trajectory of a point in the set never leaves it. Here is a more traditional example of a dynamical system.

The Lorenz Mask

The next pocket program is also a dynamical system insofar as a rule inside an eternal loop (lines 60,70, and 80) moves the point (X,Y,Z) initially set on line 10, along a trajectory. This time, however, the dynamical system is frankly deterministic; there is no call to a pseudo-random number generator because there is only one rule to choose from.

```
 5 REM LORENZ
10 X=0.9:Y=0.1:Z=0:CLS
20 READ XO,YO,UX,UY,E,N,D
25 DATA 120,32,1,-1,.05,-50,.02
30 XR=XO-N : XL=XO+N
40 YP=YO+Y*UY:REM LOOP HERE
50 PSET(XL+(X-E*Z)*UX,YP)
55 PSET(XR+(X+E*Z)*UY,YP)
60 X1=X+10*(Y-X)*D
70 Y1=Y+(28*X-X*Z-Y)*D
80 Z1=Z+(X*Y-8*Z/3)*D
90 X=X1:Y=Y1:Z=Z1:GOTO 40
```

It is a 3-dimensional dynamical system because the point being moved has 3 components. This raises the problem of how to represent 3-dimensional data in the two dimensions of a picture. On a slow computer, with a coarse-grained, monochrome video display, the best way to do this is by means of stereo-pairs. On faster machines with more advanced graphics primitives there are other ways of achieving the illusion of 3-dimensional plasticity. Here is the list in the order of presentation in Math 198: perspective, depth-cueing, motion, z-buffering, shading.

Stereo-pairs are viewed with the help of devices which insure that your right eye sees one image, while your left eye sees an image of the same scene from a slightly different angle. Our visual system is very forgiving. It is not necessary to compute these two views very accurately, which would slow things down even more. Here we use a small shear, lines 50 and 55, to approximate binocular vision. A shear is a distortion which moves a rectangle to a parallelogram without changing its base or altitude. Think of a stack of playing cards pushed uniformly to one side.

In the absence of helpful optical devices it is easier to cross your eyes. So, shift the right-eyed view to the left, and vice versa. Cross your eyes by focusing at an object, e.g. pencil tip, roughly halfway from your nose to the screen. Wait until you see three rather than four fuzzy images. Then wait until the middle one comes into sharp focus. On the printed page, the images are smaller and closer together. Here the view for the right eye is on the right. These can be viewed unaided by focusing your eyes at infinity (not crossed). One way of achieving this is to place your nose right up to the image until your eyes are relaxed (unconverged, unfocused). Then move the page back slowly until the fused, 3-D image jumps into focus. To reverse right with left in the program, change the sign of the nose offset, N, or the eye-shear fraction, E, but not both.

The shape you see developing is called the Lorenz Mask and it is a very popular example of a strange attractor. Strictly 2-dimensional dynamical systems do not need the complication of stereo-viewing. On the other hand, they don't have strange attractors. The Lorenz is also a favorite character in Nonlinear Mechanics because the rule that reads the velocity at a point is not a linear function of the coordinates of the point.

The iteration loop begins at line 40. This program uses both world and screen coordinates. On line 40, the vertical screen coordinate, YP is obtained by adding the fraction Y of the vertical unit UY to the vertical origin YO. This interprets the world coordinate Y as a fraction (proper and improper) of the fixed vertical displacement. It is an example of the axonometric projection from 3 to 2 space. Such a projection may be described informally as follows. Draw three line segments in the picture plane, the x,y,z-axes, from a common

point, the origin. The axonometric image of a point (a,b,c) in 3-space is located at the end of a path that begins at the origin O, moves along the x-axis to point A for which the segment OA has the ratio a : 1 to the axis, then moves parallel to the y-axis a ratio b : 1 , then a ratio c : 1 parallel to the z-axis.

In lines 50,55, the horizontal displacement from the left-origin, respectively right-origin, is computed. Starting from the true displacement, X, as seen by the cyclopean eye (in the middle of your forehead), which is the same for both eyes at the point you are looking at, it becomes progressively greater as the point recedes into the background, ie., as the z-coordinate becomes greater.

In lines 60-80, the world point (X,Y,Z) is moved the small fraction D along a displacement, the velocity vector, which itself depends on the current position. The reason this simplest of all numerical integration techniques is perfectly adequate here is that the dynamical system has a strong attractor. Even if at each step the computed point moves to another, nearby trajectory, it will converge to the attractor anyway.

The Third Dimension

Effective management of the depth illusion is a major theme in Math 198 since one aspect of the course is to "perceive" 4-dimensional reality in its 3-dimensional "shadows." When only one method is used to indicate depth, any momentary ambiguity spoils the illusion, blinking while looking at a Necker cube, for example. So a second method, depth-cueing for example, is good insurance. This means that points further back are drawn more dimly than those in front. The Apple IIgs has 4-bit color pixels; that is, a pixel may be assigned one of 16 shades of gray proportional to the distance of the point from the viewer. Cary Sandvig's graphics package has a number of pixel-operations built into it. A pixel-operation is simply the ability of storing the result of a logical operation between the color number about to be assigned to a pixel and the number that is already there. Standard point plotters just overwrite the old pixel. The pixel function that replaces two numbers by their maximum is the one used here to simulate depth-cueing. Another pixel function, the exclusive-or function, is used for simulating separate pixel planes, for example a cursor that can pass through a picture without alteration.

The next, mechanically more demanding object, is a depth-cued line. This is useful to improve the legibility of a rotating cube, or hypercube. For this we switch to ISYSForth because it compiles code that is fast enough to rotate simple wire-frames composed of user-built line-segments. That is, the student learning the Bresenham line drawing algorithm can implement it in Forth together with personalized pixel operations. Very simple polygonal surfaces can be rendered in a way that simulates Z-buffering by programming scan-lines that fill triangles. The pixel operations make it possible to program a limited but recognizable texture map simulation.

Most Math 198 students are eager to skip over these details and move to the Iris where such graphics are library primitives. Some, however, take the opportunity to become more deeply involved with graphics primitives and, for their class project, produced an ML and C based graphics package for their own personal 386 based home computers.

This is as good a place as any to defend Math 198 against the charge of being a course which teaches computer engineering without a license. There is mathematics in every-

thing, and the study of anything sufficiently interesting to bright students becomes mathematics when the epistemological approach itself is analytical rather than merely practical and goal oriented.

Hypergraphics on the Iris

At this point in the course interest and attention begin to bifurcate. Cellular automata and Mandelbrot sets can still be done on the Apples using the (by now familiar) languages of &-GRAFIX, CSL and ISYSForth. Some students now experiment with possible projects using these methods. All students migrate to the Iris lab for a 3-week introduction to geometrical graphics. We next discuss the source code for illiSnail. This example is typical of the RTICAs we use for anything from a 30 minute hands-on demo to a two week summer workshop for math teachers.

The Curriculum

In the first lab session the students learn to control the animation. For illiSnail this entails flying through a Möbius band so stretched that its boundary lies in a plane. It looks vaguely snail-like. I first saw a wire mesh model of this surface hanging from the ceiling of Bernard Morin's office in Strasbourg. According to Larry Siebenmann (1982) the engineer Michel Pintard made a wire model like this in the 1930s. Pintard had studied topology with Hadamard. I was deeply impressed by the beautiful, computer generated 16 mm film of this surface made by Dan Asimov and Doug Lerner (1984) at Lawrence Livermore National Laboratory with a Cray-1 supercomputer. But I had to wait for the Iris 4D to write a real-time, interactive computer animation. In fact, this surface is an interesting example of a significant class of ruled, minimal surfaces in spaces with elliptic geometry, such as the 3-sphere in 4-space. The RTICA is capable of generating several other significant surfaces, such as Steiner's Roman Surface the Clifford Torus and Lawson's Minimal Kleinbottle, a portion of which constitutes Brehm's Trefoil Knotbox. The exercises are listed at the end of the chapter. (See also Color Prints 11 and 12.)

In the second session, students learn the geometry of these surfaces and their homotopies. The third session is a survey of the internal operation of the program. It serves as an introduction to geometric computer graphics. In the fourth and last common session, all students use the RTICA (or minor modifications of it) to produce a brief (1-2 minute) animation by recording their manipulations in a script file. This is the "lab-report" recording the results of their exploration into hypergraphics.

The Program

The program listed here is actually a condensation into a single file of several RTICAs of graded difficulty and sophistication. It was designed on a Personal Iris 4D/25G using Irix 3.3.1. It was recompiled on some other systems using Irix 4.0 to increase our confidence that, with some minor tinkering, this code will run on any Iris. There are still a few bugs in it, only some of which are intentional. It is an old Navajo custom to weave an error into every blanket to forestall the temptation to imagine the work to be perfect.

For the sake of brevity we omitted several useful and instructive features, in particular Chris Hartman's script writer and object maker. Both produce textfiles. An illiScript

captures the key-presses and so recreates the animation by reading it back into the same RTICA. An illiObject is the display list for a particular stage of a homotopy of the surface. It is intended to be used by a more elaborate and sophisticated surface viewing program. The reader may obtain the source code for illiSnail from the author. Videotapes showing solutions to the exercises below, and other experiments, may be obtained from the NCSA (Francis, Chappell, & Hartman, 1994).

This program is written in "vanilla-C," using only a few of the most useful functions in the Iris graphics library. No attempt was made to write the program in exemplary C. Nor does its style conform to standard rules of "pretty printing." In Math 198 we treat programs like proofs, in which the visual space occupied by a symbol is roughly proportional to its mathematical importance. The program is meant to be studied and "unpacked" slowly before it is modified or rewritten. In particular, students are encouraged to practice using the editor of their choice on the Iris to rewrite the code in their favorite style. One time, a student translated a C-program into Fortran in order to understand it. Ironically, it had been translated from Fortran to C for the purpose of teaching it to the class. Common programming problems often have more than one solution in C. For any given problem, the absolutely optimal or most elegant solutions was generally not used, mainly because I probably don't know it. The student is certainly welcome to teach the teacher a trick or two. So, without further apologies, here is a print-out of the code which I shall document with just sufficient detail to be of profit to anyone with access to an Iris computer with the standard ANSI-C compiler and the shared libraries in its directories.

```
/* George Francis, Glenn Chappell and Chris Hartman */
/* Mathematics Department and NCSA */
/* (C) 1991 Board of Trustees */
/* University of Illinois, Urbana, Illinois 61801 */
/* e-mail: gfrancis@math.uiuc.edu */
/* descendant of knottox.c, mobius.c, cubevert.c, fly7.c */
/* this version 11/20/90 */
#include <gl.h>          /* graphics library */
#include <device.h>      /* device library */
#include <math.h>        /* mathematical library */

#define  MAX(x,y)          ((x<y)?y:x)
#define  MIN(x,y)          ((x<y)?x:y)
#define  ABS(x)            ((x<0)?-x:x)
#define  DG                M_PI/180.
#define  C(t)              fcos(t*DG)
#define  S(t)              fsin(t*DG)
#define  FOR(a,b,c)        for(a=b;a<c;a++)
#define  IF(K)             if(getbutton(K))
#define  SOAK(K)           while(getbutton(K))
#define  TOGGLE(K,f)       IF(K){f=1-f; SOAK(K);}
#define  IFSHIFT           if(getbutton(LEFTSHIFTKEY)||getbutton(RIGHTSHIFTKEY))
#define  PRESS(K,A,b)      IF(K){IFSHIFT{A;}else{b;}}
#define  PRES_S(K,A,b)     IF(K){IFSHIFT{A;}else{b;};SOAK(K);}
#define  LABEL(x,y,W,u)    sprintf(phrase,W,u); cmov2(x,y); charstr(phrase)
#define  DOT(aa,bb)        (aa[0]*bb[0]+aa[1]*bb[1]+aa[2]*bb[2])
#define  NRM(aa)           fsqrt(DOT(aa,aa))

int grnd, sndr, cube, thick, win, fly, binoc, msq, gap, brt,
    th0,th1,dth,odth,fdth,ta0,ta1,dta,odta,fdta;
float lux[3]={1.,2.,3.}, lu[3], ld[4][4], aff[4][4],
    alfa,beta,lima,amb,pwr,maus,nose, mysiz,focal,speed,far ;
char phrase[256];
float vq[8][3]; /* Kummmush cube in main */
int fr[]= {0,1,5,1,3,7,3,2,6,2,0,4,5,7,6,4};
long znear,zfar;
```

```
deFaults(){int ii,jj; /* in the key of Z */
/* surface patch */
   alfa = 2; beta = 1; limA=0.; gap = 1;
   th0 = 90 ; th1 = 269; dth=18; fdth= 6;
   ta0 = 90 ; ta1 = 270; dta=18; fdta= 6;
   dta = dta; dth = dth;
/* flags */
   rndr=1; cube=0; win=2; msg=1; thick=4; binoc=0;
/* flying  */
   maus=.01; speed=.04; fly=0; mysiz=.02, focal=2., far=9.5;
/* rendering*/
   grnd = 0; brt = 255; amb= .3; pwr= 16.; nose=.06;
/* reset the affine matrix */
   FOR(ii,0,4)FOR(jj,0,4)aff[ii][jj]=id[ii][jj];
/* and put the object into the background */
   aff[3][2]= -4.2;
/* z-buffer and depthqueing parameters */
  zNear= 0xf; zFar = 0x7fffff;
}
DeFaults()   /* if you want another set of them */
{
argumen(s(argc,argv) int argc; char **argv; /* Pat Hanrahan, 1989 */
     {while(--argc){ --argv; if(argv[0][0]=='-') switch( argv[0][1]){
     case 'w': win  = atoi(argv[1]); argv++;argc--;break;
     case 'g': grnd = atoi(argv[1]); argv++;argc--;break;
     case 'p': alfa = atoi(argv[1]);        /* Moebius band a=2,b=1 */
               beta = atoi(argv[2]);        /* Clifford torus 2   2 */
               argv += 2; argc -= 2;break; /* Knotpox        2   3 */
     case 'l': lux[0] = atof(argv[1]);      /* light source direction */
               lux[1] = atof(argv[2]);      /* 0. 0. 1. for headlight */
               lux[2] = atof(argv[3]);      /* 1. 2. 3. is default    */
               argv += 3;argc -= 3;break;
\.   * to add commandline arguments ad libitum  mimic the syntax */

int paint(lmb,dog,cat)float lmb; int dog,cat,{ int rr,gg,bb; float spec;
     rr = dog;                                      /* Chris Hartman, 1991 */
     gg = abs(cat - 128) + 64;
     bb = brt - cat;
     lmb = MAX(lmb,amb);
     spec=MIN(255,brt*( 1. + pwr - pwr*lmb));  /* Ray Idaszak, 1989 */
     rr = MAX(lmb*rr, spec);
     gg = MAX(lmb*gg, spec);
     bb = MAX(lmb*bb, spec);
     return( (bb<<16) + (gg<<8)+rr);
}
surf(vv,th,ta) float vv[3]; int th,ta;{float ll, xx,yy,zz,ww,xl;
       ll  = C(lima) - S(lima)*C(ta);  /* limaconic homotopy */
       xx  = C(alfa*th)*C(ta)*ll;      /* Sudanese Moebius Band *.
       yy  = S(alfa*th)*C(ta)*ll;      /* Sue Goodman & Dan Asimov, ca 1980*/
       zz  = C(beta*th)*S(ta)*ll;      /* Larry Siebenmann, 1982 */
       ww  = S(beta*th)*S(ta)*ll;      /* Blaine Lawson, 1970 */
       xl =  .7071*(xx-ww);            /* nearly polar  projection */
       ww =  .7071*(xx+ww);
       vv[0] = xl*7/(10+9*ww);
       vv[1] = yy*7/(10+9*ww);
       vv[2] = zz*7/(10+9*ww);
}
drawsurf(){int th, ta, dog; float vv[3], aa[3], vo[3], nn[3], lmb;
  for(th=th0; th < th1; th += dth){
    bgntmesh();
        dog = 255*(th-th0)/(th1-th0); cpack( paint(.8,dog, 0));
        surf(vo,th,ta0); v3f(vo);                       /* first vertex */
        surf(aa,th+dth-gap,ta0); v3f(aa);               /* first rung */
    for(ta = ta0+dta; ta <= ta1; ta += dta){
       surf(vv,th,ta ); /* normal on vo aa vv */
       nn[0]=(aa[1]-vv[1])*(vo[2]-vv[2])-(aa[2]-vv[2])*(v .[1]-vv[1]);
       nn[1]=(aa[2]-vv[2])*(vo[0]-vv[0])-(aa[0]-vv[0])*(v  2]-vv[2]);
       nn[2]=(aa[0]-vv[0])*(vo[1]-vv[1])-(aa[1]-vv[1])*(v  0]-vv[0]);
       lmb = DOT(lu,nn)/NRM(nn); if(lmb<0.)lmb = -lmb;
       cpack( paint(lmb, dog,255*(ta-ta0)/(ta1-ta0)));
       v3f(vv); surf(aa,th+dth-gap,ta); v3f(aa);
       vo[0]=vv[0]; vo[1]=vv[1]; vo[2]=vv[2];
```

BEST COPY AVAILABLE

```
        }
      endtmesh();
    }
}
drawhoop(ta,dt)int ta,dt;{int th; float vv[3];
    cpack(0x44ffff);
    bgntmesh();
        for(th=th0;th<th1-dth; th += dth){
            surf(vv,th,ta );v3f(vv);surf(vv,th,ta+dt);v3f(vv); }
    endtmesh();
}
drawcube(){ int ii; /* Steve Kommrusch, 1984 */
    RGBrange(100,100,100,255,255,255,gnear,gfar);
    linewidth(thick); depthcue(1);
    bgnline();
    FOR(ii,0,16)v3f(vq[fr ii]);
    endline();
    depthcue(0);
}
drawstars(){ int ii,jj; float vv[3],tmp[4][4]; /* Glenn Chappell, 1991
    pushmatrix(); pushmatrix(); /* 2 copies of the projector */
    loadmatrix(aff); getmatrix(tmp);
    tmp[3][0]=tmp[3][1]=tmp[3][2]=0; /*pure rotation */
    popmatrix(); multmatrix(tmp);
    srandom(1);  /* the stars don't change, hence the 1 */
    cpack(0xffffff);
      FOR(ii,0,1000){
      FOR(jj,0,3) vv[jj] = random()/(float)0x40000000-1.0; v3f(vv); }
    endpoint();
    popmatrix(); zclear(); /* the way it was before */
}
messages(){ /* text information as heads-up display */
    viewport(0,1279,0,1023); ortho2(-1.25,1.25,-1.,1.);
    if(!banoc){ cursor(0, cpack(0x888888); circ(0.,0.,.01);}
    cpack(grna0):0xffffff;
    LABEL(-.9, .95,"(B&C)ape (Z)ap (C)oarse (F)ine (G)ap= %d ",gap);
    LABEL(-.1, .95,"%d =(BAR)fly (B)inoc (R)ender (Q)ube (PRNTSCRN)",fly);
    LABEL(-.9,-.95,"illiSmall RTICA by George Francis, U Illinois, 1992",win);
    LABEL( .1,-.95,"  The shifted key inverts the action, usually.",win);
 * frequently used control keys */
    LABEL(-.9, .85,"(M)aus   %g",maus);
    LABEL(-.9, .80,"far cli(P)er= %.2g",far);
    LABEL(-.9, .75,"pupil near clipper (i)n (o)ut %.2g", mysiz*focal);
    LABEL(-.9, .70,"f(O cal factor %.2g",focal);
    LABEL(-.9, .65,"my s(I)ze %.2g",mysiz);
    LABEL(-.9, .60,"(S)peed  %g",speed);
    LABEL(-.9, .55,"(L)imacon %g",lima);
/* rarely used control keys */
    LABEL(-.9, .30,"(f1)ambient %g",amb);
    LABEL(-.6, .30,"(f2)specular %g",pwr);
    LABEL(-.3, .30,"(N)ose %g",nose);
    LABEL(-.0, .30,"(f5)th0 %d",th0);
    LABEL(.25, .30,"(f6)th1 %d",th1);
    LABEL(.5 , .30,"(f7)ta0 %d",ta0);
    LABEL(.75, .30,"(f8)ta1 %d",ta1);
}
keyboard(){ /* control keys */
    TOGGLE(PRINTSCREENKEY,msg) /* messages and cursor vanish */
    TOGGLE(RKEY,rndr)          /* surface vanishes */
    TOGGLE(QKEY,cube)          /* stick cube appears */
    TOGGLE(SPACEKEY,fly)       /* flying mode */
    PRES_S(SKEY,speed += .01, speed -= .01) /* accelerate */
    PRESS(ZKEY, DeFaults(), deFaults())     /* zap changes */
    PRESS(IKEY, mysiz /= 1.1, mysiz *= 1.1) /* rescale the world */
    PRESS(OKEY, focal *= 1.1 , focal /= 1.1) /* telephoto */
    PRESS(PKEY, far *= 1.01 , far    /= 1.01)   /* beware of this */
    PRESS(MKEY, maus += .01 , maus -= .01 )     /* gentler, kinder mouse*/
    PRES_S(BKEY, binoc=nose=1, binoc=1; nos-=.06) /* cross your eyes */
    PRESS(NKEY,  nose -= .01 , nose += .01 )    /* parallax adjuster */
```

BEST COPY AVAILABLE

```
    PRES_S(CKEY, dth=dta += 1 , dth=dth; dta = dta) /* coarser mesh */
    PRES_S(FKEY, dth =MAX(1,--dth);dta=MAX(1,--dta), dth=fdth;dta=fdta)
    PRESS(LKEY, lima -= 1., lima += 1. )   /* Limacons of Pascal homotopy */
    PRES_S(GKEY, gap=0 , gap=MIN(dth,gap+1)) /* between the ribbons */
    PRESS(F1KEY, amb -= .01, amb -= .01 )    /* ambient floor */
    PRES_S(F2KEY, pwr = 1. ,  pwr *= 2. )    /* specular ramp */
    PRESS(F5KEY,  th0 = MIN(++th0,th1) , th0++) /* source patch */
    PRESS(F6KEY,  th1 = MAX(--th1,th0) , th1++)
    PRESS(F7KEY,  ta0 = MIN(++ta0,ta1) , ta0++)
    PRESS(F8KEY,  ta1 = MAX(--ta1,ta0) , ta1++)
}
main(argc,argv) int argc; char **argv; {int ii,jj,dx,dy; float temp;
/* Kronecker delta for the identity */
      FOR(ii,0,4)FOR(jj,0,4)id[ii][jj]=(ii==jj)?1:0;
/* Steve Kommrusn's cube by Gray-code */
      FOR(ii,0,8)FOR(jj,0,3)vq[ii][jj]=(ii&(1<<jj))?1.0:-1.0;
/* load up deFaults */
     deFaults();
/* but use Hanrahan's argumentor */
      arguments(argc,argv);
/* normalize light the light direction */
      temp=NRM(lux); FOR(ii,0,3) lux[ii] /= temp;
 * decide on window style */
      switch( win){
      case 0:                              ; break;
      case 1: prefposition(0,640,0,512); break;
      case 2: prefposition(0,1279,0,1023); break;
      }
 * open the windows *
      winopen("ribbonball");
      RGBmode(); doublebuffer(); gconfig();
      lsetdepth 0,0x7fffff; /*tmas may be the default *
      zfunction(ZF_LESS); /* z-buffer reversed by GREATER */
/* keep trix 4.  from messing up your windows *
qdevice LEFTMOUSE); qdevice(MIDDLEMOUSE); qdevice(RIGHTMOUSE);

while(!getbutton(ESCKEY)){  /* eternal loop */
          keyboard();
 * Chappell's flyer *
          dx = getvaluator(MOUSEX) - 512; dx = ABS(dx)>5?dx:0;
          dy = getvaluator(MOUSEY) - 512; dy = ABS(dy)>5?dy:0;
          loadmatrix(id);
          if(!fly) translate(aff[3][0],aff[3][1],aff[3][2]);
          rot(-dy*maus,'x'); rot(dx*maus,'y');
          PRESS(RIGHTMOUSE,  rot(-10,'z'),  rot(-1,'z'))
          PRESS(LEFTMOUSE,   rot( 10,'z'),  rot( 1,'z'))
          if(fly) translate(-aff[3][0],-aff[3][1],-aff[3][2]);
          PRESS(MIDDLEMOUSE, translate(0,0,-speed),translate(0,0,speed));
          multmatrix(aff); getmatrix(aff);
/* rotate light source */
      FOR(ii,0,3){lu[ii]=0; FOR(jj,0,3) lu[ii] += aff[ii][jj]*lux[jj];}
/* draw a frame */
      cpack(grnd?0xffffff:0); clear(); zclear();

   if(binoc) viewport(0,640,256,768);    /* right eye is left*/
/* cross-eyed shifted stereoscope  gxf 2.29.92 with gcc flyer */
     window(-mysiz*1.25,mysiz*1.25,-mysiz,mysiz,mysiz*focal,far);
     drawstars();
     translate(-nose,0,0); multmatrix(aff);
     drawhoop(ta0,-2); drawhoop(ta1,2);
     if(rndr)drawsurf(); if(cube)drawcube();

  if(binoc){
     viewport(640,1280,256,768);  /* left eye is right */
     window(-mysiz*1.25,mysiz*1.25,-mysiz,mysiz,mysiz*focal,far);
     drawstars();
     translate(-nose,0,0); multmatrix(aff);
     drawhoop(ta0,-2); drawhoop(ta1,2);
     if(rndr)drawsurf(); if(cube)drawcube();}
  cursoff(); if(msg)messages();
  swapbuffers();
  reshapeviewport();
 /* end  while loop */ } /* end it all */
```

BEST COPY AVAILABLE

#include, #define, and variables.

The program is entirely self-contained, up to calling functions in the graphics, device and mathematical libraries. Next come a series of abbreviations for the compiler which improve the readability of the source code. The trig function macros, present here merely for notational convenience, are an occasion for discussing how to optimize real-time animations. There are still many graphics computers slower than an Iris where considerable improvement in real-time performance can be achieved by tabulating the values of transcendental functions.

For many sound but admittedly controversial reasons, we depart from standard programming practice in the matter of variable types, and other customs. The extra precision of "long" and "double" is rarely needed, and the names "int" and "float" are more mnemonic anyway. This is not, however, the place to defend or promote these departures. A reader who is offended by this rough but practical style of writing C is welcome to blue pencil my code as if it were a school-boy's effort.

deFault(), arguments()

Almost all variables which are or may someday be, interactively manipulated, or which are, or may be, used by more than one independent subroutine, are global. Their default values are assigned in a subroutine, which itself can be called repeatedly by the user during execution. There is a second set of defaults which the student can use to customize his own version of illiSnail.

The RTICA uses an exceedingly simple algorithm for reading arguments from the command line. A one letter flag announces new default value(s) for the desired parameter set. The student can easily add and subtract cases in the switch block without rebuilding the subroutine. Many years ago, when I explained to Pat Hanrahan that my students cannot spend much effort on learning input/output syntax in C, he designed this routine for them.

paint()

The surfaces the RTICA generates are all painted and lighted. That means two things. The color of a vertex is a geometrical attribute, for example a function of the surface parameters. The corresponding values of red, green and blue are attenuated proportional to the Lambert lighting model (Francis, 1987, pp. 61-64). This needs only the computation of the cosine between the normal direction and the direction of the light source. At the dim end, we clamp this attenuation at what corresponds to an "ambient" level. At the bright end, we ramp all three values steeply to pure white to give the illusion of a specular region. Later, at the cost of one 3x3 matrix multiplication per frame, we move the light source so that the specular high-light appears to be stationary as a surface rotates about some axis. This simplification of the standard Phong lighting model evolved from one which Ray Idaszak developed for the Etruscan Venus Project (Francis, 1987, Appendix). It is easy to explain and to apply, and it works quite well, especially for one-sided surfaces. Its merits and demerits are discussed elsewhere (Francis, 1991; Idaszak, 1988).

Chris Hartman mixed the present color palette in rich pastels suitable for videotaping. The meaning of the colors painted on the surface is quickly discovered once the user manipulates illiSnail to draw a fine-grained rectangular patch (see Exercise 1.1). In this subroutine the "dog" and "cat" chase each other through a color gamut as they map the surface parameter values into a mixture of red, green, and blue.

surf(vv,th,ta)

This function returns the position of the mapping.

$$v = f(\theta, \tau)$$

For another surface, the programmer need only change this function, and adjust the default parameter values. This code segment becomes less of a mystery once it is transformed into standard mathematical notation as follows.

We first map a $\pi \times \pi$ sized rectangle to 4-space, where it occupies a patch on one of the real algebraic, geodesically ruled, minimal, surfaces immersed in the 3-sphere, as described by Lawson (1970).

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \cos(\alpha\theta) \\ \sin(\alpha\theta) \\ 0 \\ 0 \end{bmatrix} \cos\tau + \sin\tau \begin{bmatrix} 0 \\ 0 \\ \cos(\beta\theta) \\ \sin(\beta\theta) \end{bmatrix}$$

We rotate the 3-sphere in the xw-plane,

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x-w}{\sqrt{2}} \\ y \\ z \\ \frac{x+w}{\sqrt{2}} \end{bmatrix}$$

and finally project this to 3-space from a point just outside the 3-sphere to avoid accidental zero-division.

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} \frac{7x}{10+9w} \\ \frac{7y}{10+9w} \\ \frac{7z}{10+9w} \end{bmatrix}$$

Multiplying [x,y,z,w] by the factor (cos (λ)-sin(λ)cos(τ)) before projection, has the effect of moving the semicircular τ-wires, for λ = 0 to 90°, through the Limaçons of Pascal, to full circles of half the diameter and all passing through the origin.

drawsurf(), drawhoop()

The surface itself is drawn as a succession of ribbons with a greater or smaller gap between them. Each ribbon, parametrized by τ, uses the triangular mesh function of the Iris graphics library. The gap is controlled by the G-key, the stepsize of θ and τ is controlled by the F-key and the C-key, which switch between a fine and a coarse mesh. The shifted F-key makes the mesh finer, the shifted C-key makes it coarser. The R-key toggles the rendered surface on and off. It is intended to switch to the wire-frame, a feature students are invited to install as an exercise. The yellow θ-ribbon, generated by the drawhoop() routine, is neither painted nor lighted to really illustrate the tmesh() syntax. The RGB-color primitive cpack(0xbbeedd) employs a hexadecimal encryption of the 3 color values. In this example, it yields bb=11*16+11=187 blue, ee=238 green, and dd=221 red, in that order.

drawcube(), drawstars()

The "unit" cube is in this RTICA for reference (its inside radius is 1) and to train the user in cross-eyed binocular viewing of the stereo images. The parameters of the binoculars can be interactively adjusted, and it is easier to use the familiar line-drawn cube than the unfamiliar surfaces to check the effect of such changes.

The cube itself is drawn as one continuous polygon following a vertex list. Recall that the hypercube is a significant actor in Math 198. Steve Kommrush, a student in my very first computer based edition of the course, left us with a beautifully simple algorithm for drawing the hypercube. As an elementary exercise in modifying illiSnail, the student is invited to turn the 3-cube into a 4-cube which can be rotated in all 6 planes, and thus implement Tom Banchoff's (1977) classic visualization of the hypercube.

Originally Glenn Chappell's stars were an amusing experiment, but their presence really helps keep one's sense of position while navigating the labyrinthine interior of the surfaces.

messages(), keyboard()

The next subroutine displays messages on the screen, for example the current value of parameters, and which keys to press to change them. The key-presses are interpreted by the next subroutine in one of three styles. Togglers alternate between two states. Cyclers are more sophisticated versions of this and operate like the buttons on a digital wristwatch. As students run out of keys to program, cyclers become popular despite their confusing logic. Since the keyboard is meant to be read between each frame, pressing a key can be interpreted incrementally as an "accelerator." The shifted key reverses the direction of change in the parameter. In some cases the changes are naturally big steps and one wants to force the user to think between presses. For this purpose we "soak" the key, so that it must be let up to take effect.

There are, of course, many other ways of controlling an RTICA; pull-down menus with sliders and buttons are the most popular. I invite all of my students to compare the effort and reward of a heads-up display with two handed pilotry favored by flight-simulators to

the alternative of controlling everything with the mouse and having verbose menus interrupt the animation. Soon most agree that ten-fingered users quickly learn do many things automatically and together, without a need for distracting writing in the field of view. The messages, and even the mouse cursor, can be turned off with the function key marked "Print Screen." I often joke with ardent defenders of pull-down menus and slider-bars that I would not care to be a passenger in a commercial jet flown by a pilot clicking a mouse to select control values from a pull-down menu.

main(argc, argv)

This brings us to the main block of the program which consists of a setup sequence and an eternal loop from which one can escape with the escape key. Students are initially discouraged from changing this part of the program because it contains the hardware specific calls in the correct order. However, an understanding of its operation in general terms helps one to perform the experiments and to interpret the sometimes baffling outcomes.

In the setup, the identity matrix is built with a Kronecker delta defined in terms of the ternary operator of C. This prepares the student for its use in Steve Kommrush's very clever construction of the unit cube. Next, the default values are assigned to the parameters, and perhaps modified by Hanrahan's routine. For example, a new light source, perhaps a headlight for the flier, can be given on the command line. Its unit direction is calculated by the program. The student might build in several lights, or a local light source as an exercise.

The current program is simplified to work properly only with a full Iris screen of size 1280x1024. Indeed, it must be suitably adjusted to work on the small Indigo screen. The command line choice of three window styles is a start in this adjustment. On the other hand, if a smaller window is needed without recompiling, execute this Unix line:

```
iris% illiSnail—w 0
```

Now we are in the loop. The mouse-syntax in this RTICA is an adaptation of that invented by Glenn Chappell in his much more sophisticated geometrical viewing program, illiFly. The intention there as here is to give the illusion of piloting a small space capsule in and around a mysterious topological object in empty space. The capsule can move forward and backward, and orbit sideways around the object. The porthole can change its focal length for wide angle or narrow angle viewing, and the entire world can change its apparent size relative to the capsule. All this is actually a plausible rationalization of the effects one can achieve using the Iris graphics library primitive for perspective projection.

At the heart of every RTICA is some way of coupling the motion of the mouse with motion in a subgroup of geometric transformation of the world coordinates. The one used here has evolved through many years of student and instructor experimentation, and is one of several we encourage new students to improve upon. Once all the function specific to the Iris graphics library are translated into standard multilinear algebra, the present version can be shown to be both obvious and nearly optimal. However, we cannot do that here. A definitive exposition of these matters is in preparation (Francis, Chappell, & Hartman, 1994).

In broad terms then, a displacement of the mouse from the center of the screen (marked by a gray bullseye) is translated into a small modification of an affine transformation of 3-space. Recall that the Iris geometry pipeline operates as if the homogeneous coordinates of each vertex are multiplied by a succession of 4x4 matrices. At any given moment, this

succession may be associated into a product of just two matrices. The first represents a member of the 3-dimensional affine group, which is a semi-direct product of GL(3,R) and R^3. (For practical purposes, think of the Euclidean group of rotations and translations.)

The second matrix represents a projective transformation which expresses linear perspective. The Iris graphics library takes a resolutely pre-Copernican view, placing the eye at the origin of the world coordinate system, and looking "backwards" into the negative z-direction. A rectangular window is placed a positive distance from the eye, and everything visible is clipped to lie in the frustum of a cone between the projection plane and a far clipping plane. There are two keys in illiSnail which control the projection matrix. The O-key changes the focal length of the view. You may think of a zoom lens. Increasing the focal parameter has a telephoto effect; decreasing simulates a wide angle lens. The latter is useful for viewing the inside of the tunnels formed by the surfaces.

In order to fly through these tunnels one has to eliminate the effect of the frontal clipping plane. The I-key does this without changing the linear perspective or the area occupied by the object on the screen. On the other hand, pressing the O-key and the I-key together, changes only the scale of the viewing window, without changing how near to the eye or the object the clipping plane is located. As you fly closer to an object, it is possible with these controls to slice frontal windows into the surface for looking in, or shrinking your apparent size so as to fly around inside.

The two states of the rotor, toggled by the space-bar and echoed on the message board, are called "flying" and "orbiting." In the former state, the axis of rotation is through the observer. Thus the space pod moves to where the mouse cursor is pointing. In the latter, it passes through the object and it appears to turn in the direction of the mouse movement as is customary for trackball rotors (Francis & Kauffman, 1994; Hanson, 1992). Press the middle mouse button to move forward at the speed adjusted by the S-key. Shift-mouse reverses the direction, while shift-S decreases the velocity. The sensitivity of the mouse can be changed with the M-key.

Binocular vision is induced, approximately, by shifting the entire scene to one side and the other before projection. For cross-eyed viewing, toggle the B-key: The right image is sent to the left viewport and vice versa. The "nose" parameter, on N-key, adjusts the binocular parallax, so that a negative value produces stereopairs for parallel viewing. Stereopairs help the user to discriminate certain surface features more accurately, and to discover programming errors during code modifications. We do not recommend crossing your eyes for any length of time.

Exercise 1.1

Here are 3 easy experiments to perform on the 3 surfaces in illiSnail. The F5—F8 keys change the range of the two surface parameters. Press the shift-key and the F7, F8 pair simultaneously to retract the Möbius band to its more familiar position of a ribbon with half-a-twist in it. Note how pressing the F and C keys switches from a fine to a coarse grained triangulation of the surface. The shifted F-key refines the triangulation, the shifted C-key coarsens it. Be aware that key presses are not buffered in a queue. All keys are polled after each frame. So if a frame takes a while, the key action is slow. In this way, the visible effect of an action is the confirmation that a key has been pressed, and no inexplicable sequence of queued up actions can happen when no keys are pressed. Retracting the other surface parameter (shift F5 F6) yields a rectangular patch which is good for studying the color scheme.

Exercise 1.2

Now press all four keys, F5–8,to restore the Möbius band and stretch it out so that its border becomes a plane circle and the "diameters" are again semicircles. If you are in a hurry, the Z-key zaps all changes and restores the original settings. Hold the L-key down and watch these semicircles close to full circles. The border of the Möbius band shrinks to a point, producing the cross-cap model of the real projective plane (Banchoff, 1978; Francis, 1987, Ch. 5). The G-key controls the width of a gap between successive meridional strips that make up the surface. Shift-G zeros the gap. Note how these ribbons are Gouraud shaded in one direction, but not the other. This improves binocular convergence as well as simplifying the code.

Exercise 1.3

For the third experiment provide the flier with a "headlight" by executing

```
iris% illiSnail–u 0.0 0.0 3.0
```

from the Unix command line. Next, rotate the snail, switch to flying mode (space-bar). Try to fly through the twisting tunnel without sliding through the walls. Once inside the snail shell, you may wish to slow-down (S-key) and widen your field of view (O-key). Release the mouse button to stand still and look around.

Exercise 2

Now switch surfaces.

```
iris% illiSnail–p 2 2–u 1. 2. 10.
```

This yields a (nearly) stereographic projection of the Clifford Torus from the round 3-sphere to flat 3-space. Repeating the first experiment demonstrates how the torus may be regarded as a closed, two-sided ribbon with one twist in it, stretched out until the edges come together along a circle. Flying through both holes of a torus is predictably easy. Exploring the limaçonic homotopy meaningfully here is more of a challenge. Note that Hanrahan's subroutine can change more than one case of default parameters. We installed "headlights" too.

Exercise 3

```
iris% illiSnail–p 2 3
```

The final surface is the most difficult to understand. The first experiment applied to this surface shows how a Möbius band spanning a (yellow) trefoil knot has 3 half twists. Bending the knot so as to form a triple-circle (think of the knots that garden hoses tend to form) brings 3 sheets of the surface together along the same curve. This 2-dimensional cell complex is a smooth realization of what Ulrich Brehm (1991) calls a knotbox. If you succeed in flying through this object, your trajectory will be a trefoil knot. A reader initiated into the topological mysteries of knot complements will recognize this complex as a standard spine of the complement of the trefoil knot in the 3-sphere.

Bonus Exercise

A rewarding programming exercise would be to enable the user to control the values of the α and β parameters interactively, say on A and B keys. This way one can observe the twisting of the band and the transitions between these three surfaces (and many other surfaces) more conveniently.

The software discussed here and documentation for running on your own Silicon Graphics computer is available through anonymous ftp from the author (gfrancis@math.uiuc.edu) by executing

```
iris% ftp 128.174.111.12
```

and logging in as anonymous.

References

Asimov, D. & Lerner, D. (1984). The Sudanese Möbius Band, 3 min. videotape in *SIGGRAPH Video Review, 17.*

Banchoff, T.F. (1990). *Beyond the third dimension: Geometry, computer graphics, and higher dimensions.* New York: W. H. Freeman.

Banchoff, T.F. (1977). *The Hypercube: Projections and slicing* [9.5 min narrated, color film]. Chicago: International Film Bureau.

Banchoff, T.F. (1978). *The Veronese surface* [12 min. color film]. Providence, RI: Mathematics Department, Brown University.

Barnsley, M.F. (1988). *Fractals everywhere.* New York: Academic Press.

Brehm, U. (1991). Videotaped interview at the Technische University Berlin.

Brisson, D.W. (Ed.)(1978). Hypergraphics, visualizing complex relationships in art, science, and technology. *AAAS Selected Symposium 24.* Boulder, CO: Westview Press.

Conway, J., Doyle, P., & Thurston, W. (1991). Geometry and the imagination, Mathematics 199 at Princeton. *Research Report GCG27.* The Geometry Center, 1300 So. 2nd St., Minneapolis, MN 55454.

Dewdney, A.K. (1988). *The armchair universe.* New York: W. H. Freeman.

Francis, G.K. (1987). *A topological picturebook.* New York: Springer-Verlag.

Francis, G.K. (1990). The snailhunt. To appear in A. Bowyer (Ed.), *Mathematics of surfaces IV; Conf. proc. of the Inst. Math. Applic.*, Bath, England.

Francis, G.K. (1991). The Etruscan Venus. In P. Concus, R. Finn, & D. Hoffman (Eds.), *Geometric analysis and computer graphics.* New York: Springer-Verlag.

Francis, G.K. (1992) *Visual mathematics, Part I: Experimental arithmetic; Part II: Hypergraphics; Part III: Geometrical graphics, 1991.* Class notes and lab manuals. : UpClose Printing Copies, Champaign IL.

Francis, G.K., & Kauffman, L.H. (1994). Air on the Dirac strings. In W. Ahikoff, J. Birman, & K. Kuiken (Eds.), *The mathematical legacy of Wilhelm Magnus.* Contemp. Math Series, Amer. Math. Soc., Providence.

Francis, G.K., Chappell, G., & Hartman, C. (1994). *A Post-Euclidean walkabout.* In the CAVE virtual reality theater (VROOM), at ACM SIGGRAPH 94, Orlando. Mosaiac documentation available on Internet from the National Center for Supercomputing Applications, U. Illinois.

Gunn, C., et al. (1990). Undergraduate research experiences, summer 1990. *Research Report GCG19*, The Geometry Supercomputing Project, U. Minnesota, Minneapolis.

Hanson, A. (1992). The rolling ball: Applications of a method for controlling three degrees of freedom using two-dimensional input devices. In D. Kirk (Ed.), *Graphics gems III*. New York: Academic Press.

Hanson, A., Munzner,T., & Francis, G. (1994). Interactive methods for visualizable Geometry. In special (July) issue of *IEEE Computer*, edited by Arie Kaufman.

Hirsch, M.W., & Smale, S. (1974). *Differential equations, dynamical systems, and linear algebra*. New York: Academic Press.

Idaszak, R.L. (1988). A method for shading 3D single-sided surfaces. *IRIS Universe*, pp. 9-11.

Illyes, R.F. (1988). *ISYSForth*. Champaign, IL: Illyes Systems.

Lawson, H.B., Jr.(1979). Complete minimal surfaces in S^3. *Ann. Math.*, *92*(2), 333-374.

Marden, A. (1991). Director. *Undergraduate research experiences, summer 1991*; Research Report GCG33, The Geometry Center, 1300 South 2nd Street, Minneapolis, MN 55454.

Sandvig, C. (1990). *GS.AMPER.NEW: A 65816 ML graphics package for the Apple IIgs*. Technical Report, UIMATH.APPLE Lab, Mathematics Department, University of Illinois, 1990.

Siebenmann, L. (1982). Le Gobelet de Möbius. [privately circulated script] Brouillon, France.

Chapter 14

A Syllabus For Scientific Visualization

ALEX PANG

What is Scientific Visualization?

Webster defines visualization as the process of seeing or forming a mental image. The panel report (McCormick, DeFanti & Brown 1987) to NSF regarding the initiative on Visualization in Scientific Computing provides a qualifier and defines scientific visualization as "...a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen...visualization is a tool both for interpreting image data fed into a computer, and for generating images from complex multi-dimensional data sets. It studies those mechanisms in humans and computers which allow them in concert to perceive, use and communicate visual information." As these definitions imply, there is a strong visual component involved. Hopefully, with the insight gained by being able to see, one can then form or understand ideas or abstracions which may lead to new scientific discoveries. Since the goal is to facilitate understanding, why limit the process to the visual channel? In fact, a more general definition, which is consistent with the original spirit and intent of visualization, may include sonification or audiofication of data that maps data to sound parameters in order to complement the visual inputs to the human brain. In addition, one may also take advantage of tactile feedback to enhance understanding through interaction and manipulation of data models with virtual reality interfaces. We shall adopt this broader definition of visualization to include different tools and feedback mechanisms that help us understand the subject under investigation.

Scope of Scientific Visualization

A thorough coverage of scientific visualization includes a complete treatment of each step in the visualization pipeline. This pipeline includes data gathering, processing, display, analysis and interpretation. Each of these steps are operating and transforming data into different forms and representations. The tools used in these operations encompass signal processing, computer graphics, image processing, graphical user interfaces, and multi-variate analysis to name a few. In addition, the tools used at both extremes of the pipeline tend to be domain dependent and often require intimate knowledge of the nature of the data sources. Obviously, such a coverage is beyond the scope of a typical school term. Thus, this chapter will concentrate on the standard tools that are commonly found in visualization systems. Do remember that visualization has a synergistic relationship with a rich set of fields including those from the different applications of scientific visualization. Therefore, keep an eye out for those visualization tools that extend beyond its originally intended application.

Rationale of Syllabus

The course material outlined below is organized into a set of core topics and a set of related topics. The core topics are intended to provide depth and basic preparatory skills to carry out visualization tasks. On the other hand, because visualization is a dynamic and growing field, the core topics are complemented with related topical breadth subjects to cover application areas and highlight the latest developments in the field. Together, these topics should provide the students with the basics and a well rounded perspective of visualization. Below is an outline of the proposed syllabus.

Core Topics:
- Data Characteristics and Types
- Data Transformations
  - Sampling
  - Quantization
  - Fourier Transform
  - Noise and Filtering
  - Registration
  - Interpolation
  - Image Enhancement
  - Feature Extraction
  - Dimensionality Reduction
  - Mapping
- Data Rendering
  - Surface Visualization
  - Volume Visualization
  - Flow Visualization

Breadth Topics:
- Computational Issues in Steering
- Visualization Packages
- Sonification and Other Input Channels
- Applications

The format of presentation for these topics will include a brief description with some examples where appropriate.

Core Topics

In operational terms, data goes through different stages of visualization where it is gathered, transformed, presented and digested. The topics in this section examine the relevant steps and issues involved. Figure 1 shows a simplified illustration of the visualization pipeline.

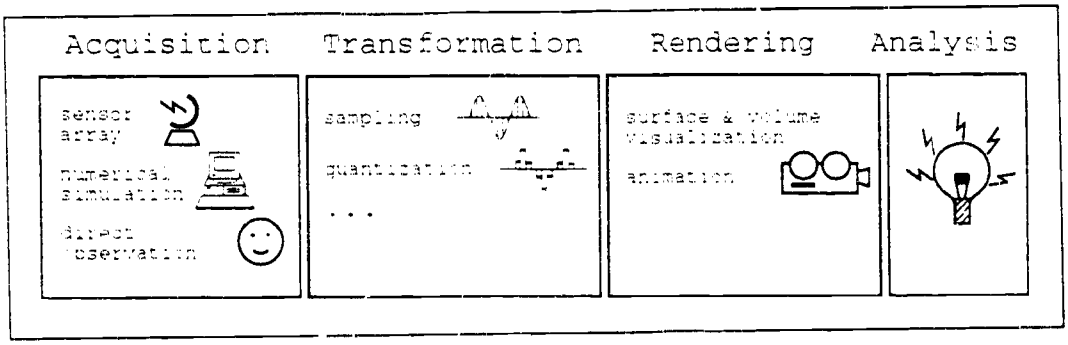


Figure 1. The Visualization Pipeline

Data Characteristics and Types

Data have different properties and characteristics and may be classified using these differences. For example, data usually represent some physical quantity which can be either a scalar or vector quantity. Scalar values may represent such properties as temperature and pressure. On the other hand, vector values may represent wind velocities. Frequently, physically related multivariate data may also be organized as a data vector. Aside from its physical interpretation, data can also be distinguished from its spatial, frequency and spectral properties. Thus, the location where data are collected, the time interval between data collection and the spectral property of the collected data all serve to characterize the data set. Sometimes the original data representation may not be sufficient to analyze the data. In such instances, one of the standard techniques in analysis is to exploit the different representations of the same data set to highlight the interesting aspects of the data set that may not be visible in other representations.

In order to properly render and present data, one must first understand the circumstances in which the data were obtained and the pre-processing that were performed. It is important, for example, to account for the calibration of sensors; noise and transmission losses; aliasing from improper sampling procedures; quantization errors while converting signals from analog to digital; and interpolation errors during reconstruction. These data characteristics can be better understood and isolated if we view them as incremental errors introduced along the visualization pipeline. At each stage of the pipeline, the output from the preceding stage becomes the input for the next stage. In this manner, we can identify four general types of data (Haber & McNabb, 1990):

BEST COPY AVAILABLE

1. Raw data.These are data that are either measured directly with sensors, obtained directly from simulations, or from first hand observations. Examples include satellite images, numerical output from a computational fluid dynamics (CFD) simulation or positions of the heavenly bodies.
2. Derived data. These are raw data that have undergone some transformation. Transformations may be as simple as direct physical relationship such as deriving electric potential from measured current and known resistance values, or they may be more involved such as inferring distance from red shift. The section on Data Transformations will discuss other forms of transformations that produce derived data sets.
3. Graphical data. Both raw data and derived data may then be converted to graphical data. Graphical data are the geometric representations and other visual cues that correspond to the raw and derived data. These may include such mappings as data value to color, opacity, elevation, graphical icons or glyphs and other geometrical, animation and perhaps sound parameters.
4. Analysis data. Hopefully, the visual representation of the data will help the investigator in the analysis task. These analyses may be an end product of visualization or it may also provide a feedback loop in the visualization process for refining data collection strategies and improving simulation model and parameters. As an example, imagine troubleshooting your car. Seeing smoke coming out from under the hood may lead you to conclude that your car has overheated. It may also warn you that your engine is not properly oiled or that it may have a leak elsewhere leading you to check other parts of the car as well.

Each step in the visualization process above involves some data corruption or transformation which must be kept track of. These transformations will be described in the next section.

Data Transformations

The visualization pipeline can be viewed as a series of modules that operate on various data sets. These modules convert data from one type to another and from one representation to another in order to highlight interesting features. As early as the data gathering stage, data is already being transformed during sampling and noise reduction operations. The data is continuously transformed all the way to the rendering stage where different view transformations provide better perspective of the same data set. This section describes some of the important data transformation steps. Note that while most of the following transformations are presented in an image processing context, the techniques can be extended to non-spatial data sets as well. Most of the references to the following transformations can be found in image processing or signal processing text books such as Pratt (1991) and Rosenfeld and Kak (1982).

Sampling

A fundamental understanding when dealing with data sets concerns how the data was sampled. Improper sampling can distort the data set and may lead to inaccurate conclusions. For example, the observation that it never rained on July 4 during the last 10 years may lead one to believe that it will not rain on the next July 4. This conclusion is not entirely

correct since climate is seasonal while weather may change on the order of a few hours or days and not on the order of years. Therefore, one should look at more recent trends instead. That is, one should watch out for the daily weather variance rather than the annual cycles. The idea that one needs to sample frequently enough to capture the rapidly changing events of the phenomena of interest was formalized by Nyquist. It is known as the sampling theory which states that "the sampling rate must be at least twice the highest frequency of interest in order to digitally represent the signal properly".

The consequence of sampling below the proper (Nyquist) rate is an effect known as aliasing. Consider the setup shown in Figure 2, where the sampler can sample at different rates and the wave analyzer outputs the perceived signal.

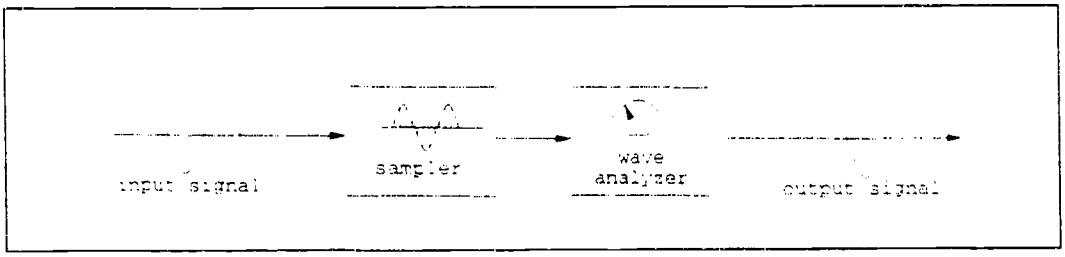


Figure 2. Data sampling

Assume that the sampler is set to sample the input signal at 60 times per second. The incoming signal may be either be simple or a composite of several signals. The perceived signal coming out of the wave analyzer will depend on whether the highest frequency of the input signal is above, below or at 30 times per second. For example, if the highest frequency of the input signal is 20 cycles per second, the entire signal can be identified properly since it is being sampled at over 40 cps. However, if the input is at 40 cps, the perceived signal will be 60-40 = 20 cps due to aliasing effects. Likewise, the perceived signal of a 60 cps input is 60-60 = 0 cps.

Aliasing effects are also manifested in the spatial domain. That is, instead of temporal aliasing, we now have spatial aliasing. For example, a scene can be digitized and represented in a 2D picture with an image resolution of NxM data points. Alternatively, the same scene can be represented with less samples resulting in a lower resolution image of nxm (where n<N and m<M). If the scene contains high frequency components such as edges and lines, those components will be undersampled and show up in the picture as aliasing artifacts usually called jaggies in computer graphics terminology.

Quantization

When an analog signal is converted to a finite precision digital representation, quantization errors are introduced. Similar errors are also introduced when a high precision signal is to be represented by a lower precision signal. For example, when continuous tone color or those represented with 24 bits are to be scaled down to an 8 bit representation, then some quantization errors are introduced. The number of bits to represent a color is known as the color resolution in graphics parlance. Aside from the color resolution, other factors are also involved in quantization errors. For example, perceived quantization errors can be minimized by optimally allocating discrete color values to represent the dominant color schemes of a scene.

BEST COPY AVAILABLE

It is also interesting to note that for human perception, tradeoffs can be made between sampling errors and quantization errors. Using the example of digital picture representations and using the same amount of information (expressed in terms of the total number of bits to represent the scene), one can use different combinations of image resolution or color resolution. For example, high frequency (busy) images are better off using high image resolution and low color resolution. On the other hand, low frequency (slowly changing) images are better off using low image resolution together with high color resolution.

Fourier Transform

Originally, the idea of this transform was to represent a complicated function with a linear combination of simpler functions. Today, Fourier transform is used in a broad range of fields from optics to numerical analysis and to medical imaging. In the realm of image processing, the 2D discrete Fourier transform, described below, offers another perspective of the picture content by transforming information in the spatial domain to an equivalent representation in the frequency domain.

$$F(u,v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) \exp\left[-j2\pi\left[m\frac{u}{M} + n\frac{v}{N}\right]\right]$$

where F(u,v) is the Fourier transformed image of an MxN input image *f(m,n)*.

Being able to analyze the image or signal in the frequency domain may provide another view of the same data content in a new light. For example, a 2D discrete Fourier transform of an image provides information about the magnitude and orientation of different frequency components in the image. The information in the frequency domain can often be exploited in various filtering schemes that operate in the frequency domain. Examples include some noise removal and image enhancement techniques. The success of the filtering schemes rely on the fact that the Fourier transform is reversible. That is, one can apply an inverse Fourier transform to reconstruct the original image or signal. One of the many practical applications of the Fourier transform relies on the projection slice theorem which claims that projecting an image onto a line at a certain orientation and then taking the Fourier transform of the projection is the same as taking the Fourier transform of the original image and then taking the values from the transformed image along a line (slice) with the same projection line orientation. If one can obtain projections at various orientations, then the information in the Fourier transformed space can be filled in. Using the inverse Fourier transform, the original image may then be constructed. This fact is exploited by computerized axial tomography (CAT) scanners. The same idea is also used by some volume rendering techniques to generate X-ray like images.

Noise and Filtering

Sensors often introduce noise which manifest themselves as high frequency components that need to be smoothed out. Different techniques are available for alleviating the effects of noise depending on whether the noise is independent or dependent on the subject being measured. Having as much information about the subject, even those of statistical nature, will aid in the removal of noise from the measurements. For instance, without knowledge of where noise are located, they can still be smoothed out by spatially averaging the values with a small sliding window. However, this low pass filtering operation also has

the undesirable effect of smoothing out some details in the data. One can do better if one knows where noise are located. Intuitively, the idea is to selectively take out the noise and replace it with something else—usually some average of neighboring values.

Registration

Just as there are different types of data, there are also a multitude of sensors. One of the tasks in visualization is to fuse these different data into something that is coherent. In order to perform multi-sensor data fusion, one must first do data registration and try to account for distortions arising from sensor placements. For example, to obtain a satellite image of the earth without cloud cover, several images obtained over a period of time, from one or more satellites must be composited together to form a mosaic image. These images must be registered properly in order to form a seamless mosaic image. A typical technique is to use certain landmarks or fiducial marks to register the rest of the data. The proper positioning and orientation of composite data sets can then be found by using correlation techniques which maximizes the match between the composite sets. Note that the same idea can be applied to non-spatial data such as speech or temporal patterns.

Interpolation

In almost every instance of data collection, data are sampled at discrete spatial locations and time intervals. If the sampling is frequent enough, the original signal can be reconstructed from the sampled values. If only a few values in the series of measurements are needed, those values may be approximated by interpolating neighboring values. Various degrees of continuity can be achieved by using different interpolation techniques. For most practical purposes, linear to cubic interpolations are sufficient. When data is being collected or generated from a spatial grid, intervening values between the grid points can also be obtained in a similar manner. Most interpolation techniques can be readily extended in dimensionality from curves to surfaces to hypersurfaces. Thus, assuming that data are sampled properly, then one can use simple linear interpolation or slightly more complicated spline interpolation to fill in missing data with a high degree of confidence.

Image Enhancement

Because of the limitations or the conditions in which sensors operate, together with some built in bias in our visual perception system, it may be worthwhile to apply certain transformations to the data set in order to enhance it. For example, an out of focus or motion blurred image may be sharpened to some extent; and details in a poorly lit image may be enhanced by contrast stretching.

Image sharpening is the counterpart of the smoothing operation. In qualitative terms, the high frequency components of the spatial frequency signal is emphasized while the low frequency components are suppressed. In operational terms, the image is convolved with a template such as the one shown in Figure 3 to emphasize areas of changes in picture values.

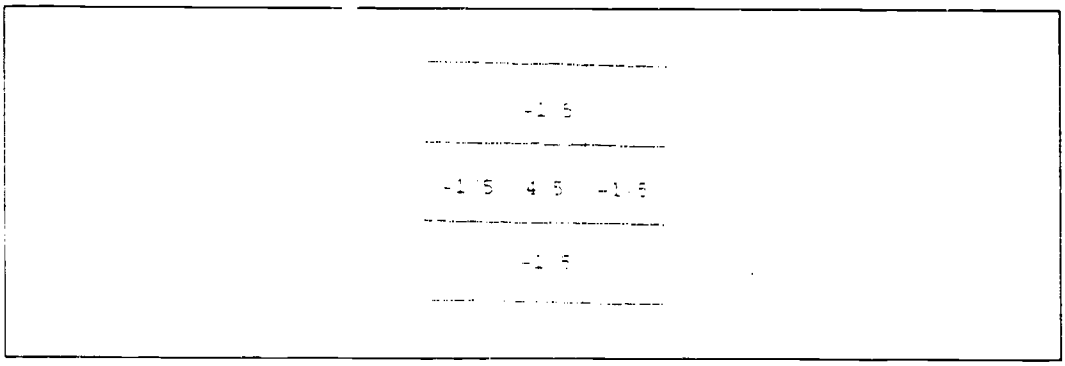


Figure 3. Image sharpening

That is, the enhanced picture $g(m,n)$ is obtained by subtracting the blurred (or averaged) components out of the original picture $f(m,n)$ for each picture element m,n using the following relationship:

$$g(m,n)=f(m,n) - 1/5\ [f(m+1,n)+f(m-1,n)+f(m,n)+f(m,n+1)+f(m,n-1)]$$

Contrast stretching is achieved by a more efficient utilization of the available quantization levels or picture values to represent the information in the image. This task is also known as histogram modification because it changes the distribution of how the quantization levels are used. The essential features of the technique is to first obtain the histogram of the image and then to flatten or equalize the histogram. An intuitive explanation of how this works is as follows: the histogram of an under-exposed image will show that dark or low quantization values are used much more than bright or high quantization values. Histogram modification will spread the large clump of dark values so that one can now resolve smaller differences in the dark region. Note that the same technique can be used to minimize quantization errors.

Feature Extraction

Often times, there is a deluge of data so that the sheer volume interferes with the analysis task. In these cases, it is helpful to simply examine interesting features rather than the original data set in its entirety. Interesting features vary from application to application and may range in sophistication from detecting the presence or absence of the feature to predicting the occurrence and tracking the location of the feature. Depending on what is being searched for, different techniques are available for extracting features from the data set. For example, a typical pattern analysis procedure will include finding contour lines and detecting edges or regions in the image. In analyzing electrocardiograms, a combination of several features such as the frequency and amplitude of a complex wave form may be significant.

Dimensionality Reduction

Like feature extraction, the techniques used here also reduce the amount of data that needs to be dealt with. The manner in which this reduction is accomplished is through

multivariate analyses where the relationship of various parameters are established. A popular method for doing this is called projection pursuit expounded by Friedman (1987) and Crawford and Fall (1990) where high dimensional data sets are successively projected into lower dimensional space such that one can maximally discriminate between sets of variables. Similarly, if variables can be related to each other in some linear fashion, then those variables can be lumped together into a single composite variable.

Mapping

Data can be prepared for rendering by mapping them to geometric primitives and visual cues. There are several options that are readily apparent. Sequential data from a single source can be plotted with points connected by piecewise linear segments or a spline curve fitted through the points. Ratio information can be illustrated using popular business graphics methods of bars and pie charts. Higher dimensional data can be represented within a single image by mapping different variables to different cues such as color, line thickness, line style, shape, and so forth. As an alternative, Chernoff (1973) proposed to use "smiley" faces or other icons to represent high dimensional data set where mapping parameters include roundness of face, expression of face, etc. Mainstream visualization typically makes extensive use of color mapping, for example use of pseudo coloring scheme to represent different temperature, pressure, excitation levels and values of other scalar variables. Vector data such as wind velocity in weather applications and CFD applications are usually represented in static images as arrows of different lengths, colors and directions. Alternatively, they can be animated. Each data point is represented as a small particle and the path of each particle is traced out over time to indicate the magnitude and direction of flow. Uncertainty in the data, be it data quality or density levels can also be mapped to different opacity levels resulting in images that contain fuzzy areas. In addition, to the visual cues, data can also be mapped to sound parameters such as pitch, volume and different musical instruments. The different mapping possibilities are countless. However, one must be judicious in selecting the appropriate mapping to avoid confusion.

Data Rendering

The most notable part of visualization is the graphical rendering of the transformed data. The rendering process itself can also be considered another stage of data transformation. In this case, transformation tasks may involve hidden line or surface elimination, color mapping, or generating geometrical data from derived data, among other things. The final output of the data rendering stage are the pictorial representations of the data. A whole spectrum of techniques is available for rendering. These may range from time critical applications to refined, post-production quality graphics used in detailed analyses.

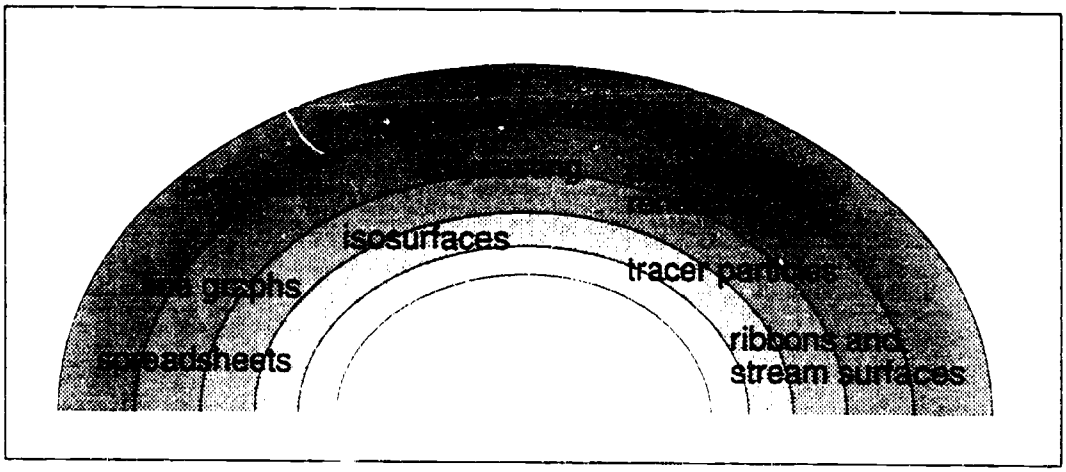


Figure 4. A spectrum of graphical rendering techniques

The techniques layed out in Figure 4 represent some of the more recent topics in visualization and are by no means exhaustive. It does illustrate the range of options that are available at one's disposal. At one end of the spectrum, the graphics tend to be "quick-and-dirty" and are often used to meet the demands for real-time interactivity. At the other extreme, the techniques are more expensive but generate more sophisticated and detailed images. Below is a brief description of some of the more popular techniques.

Surface Visualization

Physical world data that need to be visualized are usually located within three dimensional space. Traditional computer graphics algorithms have looked at the problem of hidden line and surface removal of 3D scene descriptions. In contrast to these scenes where surfaces are specified, the 3D data one encounters in visualization problems are often volumetric in nature. Data values obtained from different locations within the volume can be characterized by the type of grid that is superimposed on the environment (Wilhelms, 1991). The grid may be regular where each element in the volume is an identical box, or the grid may be rectilinear, a slightly more general grid where the distances along axes are arbitrary. Structured but non-orthogonal grids are often found in CFD experiments. These are called curvilinear grids and are often defined by warping a regular grid in space. Finally, there are the unstructured grids where sample points are distributed in some arbitrary order in space. Usually, any geometric structure present in the data set are implicit rather than explicit. For example, a CAT scan is a collection of density values within a regularly structured grid with no explicit specification of surface boundaries. On the other hand, oceanic and atmospheric data sets are usually sparse and unstructured.

Just like extracting edges from 2D images, surface information must also be extracted from the volumetric data. Surface features that are of interest include isosurface locations and their corresponding surface normals. These tasks are much simpler if the volumetric data is in a regular grid. Typically, the volumetric data can be treated as either residing in a small cube of constant value, known as voxels, or residing at corners of the grid cells. In both cases, some kind of thresholding is first performed to determine data values of interest. Others are zeroed out. Each interpretation then results in different ways of rendering. The

cells interpretation requires finding the approximate polygon surface representation of the surface of interest. The voxel interpretation works directly with the volumetric data. Let's look at both approaches a little more closely.

A popular way for converting volumetric data to polygonal representation is with the marching cubes (Lorensen & Cline 1987) algorithm. It works by identifying all possible cases on how the 8 corners of a cell can be filled with the possible surface points. Surfaces are then identified by interpolating through these corner points. This process is extended to neighboring cubes until the entire volume is processed. Once the surfaces are identified, the polygons are then sent to a standard polygon renderer. Since these surfaces were obtained by thresholding the cells to a single value, the resulting surfaces are collectively called the iso-surface. If more than one iso-surface is desired, then the process is repeated this time using a different coloring scheme for the extracted surfaces.

Unlike iso-surfaces, voxel based approaches work directly with the volumetric data. These methods are usually found in the context of medical applications where volumetric data are from CAT scans. An initial thresholding pass is performed to identify those voxels that fall within a range of values. Then a second pass is performed to render the identified voxels. There are two general methods of rendering the volumetric data. One is based on raycasting and the other based on projection.

Raycasting generates an image by casting a ray from each pixel on the display screen to the scene. The first non-zero voxel that the ray hits is the visible voxel and also terminates that ray. The shading of that particular voxel on the screen is usually determined by examining the neighboring voxels and approximating a surface normal at that point. There are also other things to consider: orthographic rays result in a more efficient implementation than oblique rays; interpolation may be necessary if the ray falls between voxels; other means of shading such as distance-based and image-based shading can be considered; and finally, extending raycasting so that secondary rays are generated (raytracing) can provide reflection and transparency effects.

Instead of shooting rays to the data set during raycasting, projection throws the data to the projection screen. As data is projected and a determination is made on the location and size of the projected image, the distance information are also recorded in order to determine visibility. Note that data can be processed in either back-to-front or front-to-back order. Shading information is derived in the same ways as those used in raycasting. Operationally, the difference between the two methods are:

1. Raycasting:
   for each pixel
      shoot ray to data set.
      find intersection and shade.
2. Projection:
   for each data point
      project onto screen and shade accordingly.

Volume Visualization

Another commonly used technique in visualization is to generate images where the entire volumetric data set contributes to the final image. The resulting images can

potentially show the internal structures instead of just surface information. Note that unlike surface visualization, there is no feature extraction step. Typically, volume rendered images tend to be more fuzzy since they employ transparency and opacity mapping in addition to color and shading mapping. Similar to surface visualization, there are two general classes of volume rendering algorithms: projection based and raycasting based. The main difference between volume and surface visualization is the use of transparency to incorporate internal structure information into the final image.

A popular method of volume rendering using the projection method is known as splatting (Laur & Hanrahan, 1991; Westover, 1990). This method assumes that each data point represents some density function where the values drop off with distance. An image is generated by projecting each data point together with some filter associated with the density function and then integrating with the information in the neighboring projected space.

With raycasting, or more generally, raytracing technique, several variations exist depending on how values are accumulated along the ray. The general idea is that as the rays are traced out into the data set, they penetrate through the volume where color and opacity information are accumulated. One method (Upson & Keeler, 1988) proposed to examine the data values as the rays penetrate the boundaries between adjacent cells. Another method (Levoy, 1988) proposed to examine the data values at equally spaced intervals along the ray. Both methods have to deal with interpolating values along the ray. Other considerations include the effect of hierarchical representation of the volume. In addition, one can also allow the ray to refract in different directions depending on the varying index of refraction as the ray traverse through the medium. For the sake of efficiency, the raytracing process is terminated when either the ray does not hit any volume element or the accumulated opacity value has reached unity. Areas where volume visualization has gained popularity include medical imaging and computational fluid dynamics. For example, taking advantage of the regular cubical structure and fixed light sources, the Heidelberg raytracing model (Meinzer, 1991) made further simplifications based on the rendering equation (Kajiya, 1986) and still produced high quality images. Another example is the volume rendering method proposed by Krueger (1991). This method is based on transport theory and offers flexible mapping between extracted features and visual parameters allowing different perspectives of the same data set.

Flow Visualization

Data sets often contain dynamic information such as velocity information or constantly changing variables with the introduction of another variable—time. Unlike structures found in static data sets such as CAT scan images, the salient points in these dynamic data sets are often found in the changing flow patterns. The most obvious way of showing flow behavior is with animation. There are several techniques ranging from traditional methods such as animation loops and color table animation to more sophisticated methods such as particle systems and ribbons. These will be discussed briefly.

A static image can also provide some sense of flow within a vector field. These images typically involve the use of hedgehogs or arrows of varying directions and lengths indicating the magnitude of the vector. More recently, Ellson (1988) proposed to pack multi-variate information together in the hedgehog display by substituting arrows with icons or glyphs. However, there are limitations on these techniques when dealing with 3D vector fields. In particular, the image will look too busy and tend to confuse the viewer. As an alternative,

Helman and Hesselink (1991) suggested that structural and topological information such as tangent surfaces or surfaces of separation can be derived from critical points such as saddle points as well as attracting and repelling foci. These curves and surfaces can then be rendered thereby highlighting features such as curls and twirls often found in CFD data sets.

A classical example where simple animation loops can highlight the important flow patterns is by taking a series of satellite cloud images and playing them over and over again in a closed loop. This technique has gained wide acceptance and is commonly found in TV weather forecast segments. The same technique can obviously be applied to data from other application domains. Another cheap method of simulating motion without doing any actual movement is with color table animation (Van Gelder, 1992). The tricky part is selecting the right color table entries such that when they are cycled through using the color indices in the displayed image. an illusion of motion is generated. Another example of displaying motion is described by (Freeman et al., 1991) with the use of steerable filters to calculate local phase changes which give rise to the sensation of motion.

Particle systems were originally proposed by Reeves (1983) to model natural phenomena. The basic idea behind particle systems can be viewed in terms of object oriented programming where each particle in the system is assigned certain properties and each particle behaves independently according to its own set of rules and possibly some interaction rules with neighboring particles. For example, fire, fireworks and grass have been modelled with particle systems by defining an initial area where particles are seeded. Each particle in this area is then assigned an initial velocity, color, life span, size and other relevant parameters. Some rules such as gravitational laws from the physical world or some arbitrary rules for some desired effect, such as decay or birth rules, are then imposed on the particles. The entire system is then activated and each particle is traced out in space and integrated over time.

The same idea has been applied to flow visualization, notably in CFD applications. In this case, tracer particles or smoke dyes can be injected at certain places and time intervals into the system. Their path and relative age are then tracked by observing changes in particle position and color. The calculations involved in determining the path trajectory are usually based on local gradients and the advection or transport of particles.

The effects of particle systems may be further enhanced by combining some ideas from behavioral animation to highlight some effects that are normally not visible such as force fields or vector fields. For example, Wejchert and Haumann (1991) showed how leaves tumble and swirl around as they are blown by the wind. That is, one does not immediately see the wind but can infer on wind speed, direction and the presence of vortices.

As a further improvement to particle systems, Hultquist (1990) proposed to use ribbons to indicate flow. Ribbons are similar to particles but come in pairs. As the particle pairs are traced out, polygons are used to tile the surface between the two particles. This idea can be generalized to a rake where particles are introduced at each of the teeth locations. The resulting set of ribbons is referred to as stream surfaces. One of the important advantage of this feature is that stream surfaces can fold or twist around thereby highlighting twisting movements that may be lost in the confusion of particles.

Breadth Topics

The topics in the following sections are meant to supplement the materials in the core topics. Depending on the interest level of the class and time availability, the topics below can be included in the same academic term. The following is by no means an exhaustive list of breadth topics and should be updated periodically as new and interesting advances occur.

Computational Issues in Steering

The materials presented so far have, to a large extent, ignored the computational issues in most of the algorithms. A closer look at the spectrum of visualization techniques will reveal that there is a tradeoff between rendering quality and speed. In this section, we will look at similar tradeoffs in achieving interactive speeds necessary for steering. Steering or navigation is a term coined to describe the ability to dynamically modify computations and their corresponding visual representations during processing. Immediately, one can identify two requirements: fast numerical solution of the mathematical models and fast rendering. Luckily, these two requirements can be pursued relatively independently.

The treatment of numerical solutions to mathematical models, typically characterized by ordinary differential equations (ODE) and partial differential equations (PDE), is a subject of its own and is beyond the scope of this syllabus. However, it is worthwhile to bring out some standard integration techniques such as Euler, Runge-Kutta and adaptive step-size methods from Numerical Recipe (Press et al., 1986) for solving ODE and Finite Differencing, Method of Lines, Finite Elements and Monte Carlo methods for solving PDE. Numerical accuracy and stability of results can be easily demonstrated through some simple homework where students can vary the integration methods and the integration time steps. A simple exercise may involve simulating a swinging pendulum or a coupled pendulum pair and then showing a graphical presentation of the system.

To address the question of obtaining fast numerical solutions, discussions will have to include various computer architectures and how both standard and innovative solution strategies can be best mapped to the hardware. Some methods, such as cellular automata approaches, have been shown to provide qualitatively similar solutions to certain PDEs with great savings in computation time. These methods essentially discretized the system in time where a small number of states that components can assume are identified. Transition from one state to another is then specified as a set of rules and often implemented as a lookup table making for very efficient and fast calculations. For applications where the sacrifice in accuracy is not tolerable, other avenues exist that allow the solution to be varied in accuracy to a satisfactory level. These may include lattice gas methods (Doolen, 1991) and other problem dependent approaches.

Like its scientific computing counterpart, the issue of fast rendering is being addressed with both specialized hardware and clever algorithms. Since the introduction of the geometry engine (Clark, 1982) there has been several computer developments specifically targeted at speeding up graphics rendering. Among the more recent include the Cube architecture by Kaufman and Bakalash (1988) for rendering voxel data sets and the Pixel-Planes 5 (Fuchs et al., 1989) for fast polygon rendering. On the algorithm front, one can take advantage of the fact that during interaction, one does not need as much detail. Thus, there are several hierarchical representations and rendering algorithms such as octree based methods used by Wilhelms and Van Gelder (1990) that can provide fast, coarse level

ERIC

displays during interaction and detailed displays later. Others take advantage of situations where a static 3D data is viewed from different positions and orientations. In such situations, the Fourier projection plane (Dunne et al., 1990) reduces the 3D rendering task to a single 2D inverse Fourier transform per view. Yet other approaches include the tiny cubes, vanishing cubes and slice methods (Nielson & Hamann, 1990).

Another issue relevant to steering is to provide such capability to scientists who do not have direct access to sophisticated hardware such as those found in a national supercomputer center. This brings up the task of providing remote interactive steering. If the compute server is some distance away from the local workstation where the graphics is displayed, one must tradeoff the amount of data to be sent through the interconnecting link as well the amount of work done at both ends. For example, both numerical solution of the models and graphics rendering can be performed at the remote site and the final images sent to the local workstation. This arrangement might be suitable if the images are small or there are only a few images to send. However, for interactive steering applications where the user needs to continuously interact with the calculation as the simulation is progressing, it may be more appropriate to have a different setup where the computational load is distributed between the remote and local machines and the overall network traffic is minimized. Other considerations include the relative merits of data compression and the cost of compression/decompression as well as some information loss.

Visualization Packages

Numerous visualization packages are available either commercially or from research institutes. Below is a list of the more popular ones and some relevant information on each. It is by no means an exhaustive list.

1. NASA: **PLOT3D**

   For NAS users:
   NAS Documentation Center, MS 258-6
   Nasa Ames Research Center
   Moffett Field, CA 94035-1000
   (415) 604-4632
   doc-center@nas.nasa.gov

   For non-NAS users:
   Computer Software Management and Information Center
   The University of Georgia
   382 East Broad Street
   Athens, GA 30602
   (706) 542-3265

2. Silicon Graphics, Inc.: **Explorer**
   Currently comes bundled with the purchase of SGI workstations.

Silicon Graphics, Inc.
P.O. Box 7311
Mountain View, CA 94039-7311

Relevant newsgroups:comp.graphics.explorer and
comp.sys.sgi

3. AVS, Inc.: **AVS4**
An international repository for materials related to AVS is being managed by the International AVS Center at the North Carolina Supercomputing Center in Research Triangle Park. Funding for the center is from a consortium of vendors.

AVS, Inc.
300 Fifth Ave.
Waltham, MA, 02154
(617) 890-4300

Relevant newsgroup: comp.graphics.avs
ftp site: avs.ncsc.org (128.109.178.23)

4. University of New Mexico: **Khoros**
Khoros is an integrated software development environment for information processing and visualization based on X11. Khoros components include visual programming language, code generators for extending the visual language and adding new application packages to the system, an interactive user interface editor, and interactive image display package, an extensive library of image and signal processing routines, and 2D/3D plotting packages.

Director: John Rasure (rasure@eece.unm.edu)
Relevant newsgroup:comp.soft-sys.khoros
ftp site: pprg.eece.unm.edu (192.31.154.1)

5. TaraVisual: **apE3**
Originally developed at the Ohio Supercomputer Center, apE has gone through several revisions and is now handled by a private enterprise.

TaraVisual Corporation
929 Harrison Avenue, St. 201
Columbus, OH 43215
1-800-458-8731

6. University of Wisconsin: **VIS5D**
VIS5D is a system for interactive visualization of large 5D gridded data sets such as those made by numerical weather models. Three of the 5 dimensions are for the physical space, one for time and the last one is organized as a vector array for different variables at a given location. Features include isosurfaces, contour line slices, colored slices, wind trajectory tracing, etc., of data sets in a 3D grid.

Principals: Bill Hibbard (whibbard@vms.macc.wisc.edu)
Brian Paul (bpaul@vms.macc.wisc.edu)
ftp site: vis5d.ssec.wisc.edu (144.92.108.63)

7. NCSA: **DataScope** etc.
The National Center for Supercomputing Applications is home to a host scientific visualization software for various platforms. Among the suite are GelReader for molecular biologists, ChemTool for computational chemistry, Isosurface Visualizer, PolyView, Contours, Image, and DataSlice.

ftp site: ftp.ncsa.uiuc.edu (141.142.20.50)

8. Research Systems, Inc.: **IDL**
IDL (Interactive Data Language) is a software system for the analysis of scientific data, a 4th generation language, and a visualization package. It is also the precursor to PV-WAVE.

2995 Wilderness Place, St. 203
Boulder, CO 80301
(303) 786-9900

9. Precision Visuals, Inc.: **PV-WAVE**
PV-WAVE comes with a set of subroutines for data reduction, filtering, transformations and analysis. A user programmable graphics interface is also available.

6230 Lookout Road
Boulder, CO 80301
1-800-447-7147

10. Intelligent Light Corp.: **Fieldview**
P.O. Box 65, Fairlawn, NJ 07410
(201) 794-7550

Sonification and Other Input Channels

Undeniably, human vision is the sensory channel with the widest bandwidth. However, there are times when even the visual channel is overloaded or the visual information too corrupted to communicate effectively. As an alternative or as a supplement to visualization, one can take advantage of other sensory inputs such as sound and tactile feedback. A simple example is the kind of information one considers when diagnosing automotive troubles. They may be visual such as looking for smoke and observing the color and amount of smoke; or they may be olfactory such as smelling for leaks; or they may be auditory such as listening to squeaks and clanks.

It has been argued that acoustic signals or the sonification of data is an attractive means of dealing with multi-variate data sets since one can map different variables to different acoustic cues such as duration, pitch and loudness (Grinstein et al., 1989). Just as visualization has icons for visual aids, sonification has earcons (Blattner et al., 1989) for audio aids. These auditory icons may take the form of warning bells and sirens or may be more problem specific. For example, in data sets where time is a dependent variable or when measurements are not taken at regular intervals, the passage of time may be marked by an earcon such as a clock tick or a drum beat.

Information can also be communicated via haptic feedbacks and by direct immersion in the data set. For example, as early as the mid 1940s, gravity-suits were used to counteract the physiological effects of acceleration. Similar G-suits can be used in a flight simulator to simulate the effects of gravity on the aviator flying at different speeds. More recently, research notably at the University of Washington, University of North Carolina, MIT, and NASA focused on virtual reality interfaces. These include the design and use of such devices as head-mounted displays and data gloves. An example where different sensory inputs are integrated into a single environment is the VIrtual Environment Workstation (VIEW) project at NASA Ames Research Center (Wenzel et al., 1990).

Applications

There are numerous application areas of scientific visualization. These application domains often share a symbiotic relationship with scientific visualization. In numerous cases, the peculiar needs of an application may serve to enrich the host of tools and techniques of visualization and help it address the needs of other fields. Below is a list of some popular application areas as well as some potential areas for future interactions.

1. ***Computational Fluid Dynamics***
   CFD is the numerical study of the flow of fluids such as air and water. A major use of CFD is the design of aircrafts. Its use often reduces the design turnaround time and minimizes the use of expensive wind tunnels. Starting with simple hedgehog diagrams for displaying velocity fields, CFD has been one of the primary pushers for the advancement of scientific visualization particularly in the area of flow visualization. Current techniques include the use of stream lines and polygons, particles, ribbons and of course, animation to aid the designer in visualizing flow patterns. Together with flow visualization, another problem that often arises in CFD applications is how visualization of volumes in irregular grids are dealt with. Aside from efficiency concerns, irregular grids also introduce some complications when interpolating values within the volume.

2. ***Medical Applications***
This is another application that is pushing the frontiers of scientific visualization particularly in volume visualization. Unlike CFD applications, the volume data found in medical applications are usually in regular grids and the information are usually static or have much longer time scales than CFD data. These factors have led to some very efficient rendering techniques of volume data from different imaging modalities such as CAT scans, positron emission tomography (PET) scans, nuclear magnetic resonance (NMR) images and ultrasound. The capability of direct volume rendering, where internal structures of the volume are made apparent by varying associated opacity values, has also been exploited in the area of surgical planning. This is where a surgeon can run through a simulation of a surgical procedure using interactive volume visualization techniques that allow him or her to do cutting and probing operations. Ideally, this prepares the surgeon for the actual operation in terms of anticipating and making contingency plans for the difficulties that were encountered during the simulation.

3. ***Molecular Modelling***
This is an area where scientific visualization has almost replaced the traditional styrofoam ball and stick models that chemists use to help understand the structure of molecules. Now, the physical balls and sticks are replaced with geometrical representations and computer renderings. One nice thing is that the user is still allowed to rotate and view the model from different perspectives. This technique has been successfully used in studies of docking molecules. A popular way of viewing the myriad of ball and stick figures is with stereoscopic projections to achieve 3D like depth perspectives. Another way of representing the models is with density clouds giving rise to images of molecules with fuzzy borders.

4. ***Climate Modelling***
Similar to CFD, both climate and weather modelling focus on changes and patterns. Traditional techniques include animation loops, iso-parametric or contour lines, and hedgehog plots. More recently, problems involving severe weather watch, and long term studies of El Nino and global warming have increasingly taken advantage of flow visualization tools. To name a few, the McIDAS system at Wisconsin and the NCAR package from Colorado provide some visualization tools that were originally targeted for environmental data.

5. ***Computer Science***
Scientific visualization has also found some application within the domain of computer science studies. Its importance is recognized in the area of performance monitoring and evaluation of parallel computers (Simmons and Koskela, 1990), program or algorithm visualization, visual programming languages and debugging tools, CAD tools for layout and design verification, and visualizing abstract objects and relations (Kamada, 1989).

6. ***Education***
An area where scientific visualization is only starting to make headways and therefore has a lot of potential for making contributions is in education. This is particularly true in science subjects where students may have a hard time grasping abstract concepts.

Already, some projects are making impacts in this area. For example, James Blinn teaches concepts in mathematics and physics in the PBS series The Mechanical Universe. Relativistic effects were also demonstrated visually by Hsiung et al.(1990). In mathematics, one can easily see how dynamical systems degenerate to chaos. There are other areas in the social sciences that can also benefit from visualization. For example, a multimedia presentation of historical or geographical events is sure to capture the attention of the students more than black and white text. As an idea, one can employ a geographic information system (GIS) to show how the boundary lines along different nations changed in history, or how continental drift manifested itself over time. Furthermore, visualization can provide decision making aids and can also be used to educate policy makers.

Where to go from here

Obviously, the material presented here is more than what can be covered in a one quarter or semester course. The recommendation is to cover basic data transformation techniques as well as data rendering techniques. As the field evolves and new techniques are invented and more applications take advantage of visualization, more topics can be added as breadth topics. When the field matures further, some of the breadth topics may even become core topics. Selection from the breadth topics is usually decided based on the interest of the class or whatever is currently exciting. At the same time, some of the core topics may be contracted. Alternatively, this could possibly be a two term course each one emphasizing either core or breadth topics.

To keep abreast in this fast growing field, below is a partial list of conferences, journals and newsgroups primarily within the US.

Conferences/Proceedings

1. IEEE Conference on Visualization, IEEE Computer Society Press
2. National Computer Graphics Association, NCGA
3. Special Interest Group on Graphics, ACM
4. Eurographics, North-Holland Publishing Co.
5. Supercomputing, IEEE Computer Society Press

Journals/Magazines

A partial list of journals that deal directly or indirectly with some of the issues in visualization can be found in:

1. *Pixel: The magazine of scientific visualization*, Pixel Communications
2. *Supercomputing Review*, Thomas Tabor, 8445 Camino Santa Fe, San Diego, CA 92121
3. *ACM Transactions on Graphics*, Association for Computing Machinery
4. *IEEE Computer Graphics and Applications*, IEEE Computer Society

5. *The Visual Computer*, Springer International
6. *Computer Vision, Graphics, and Image Processing*, Academic Press
7. *Journal of Molecular Graphics*, Butterworth Scientific Ltd.
8. *Computers in Physics*, American Institute of Physics

Newsgroups

Sites with news feed from these groups can access the respective postings. It is suggested that you start with the following set of newsgroups and prune the list down later as you discover that some are too specific for your needs, while others are not too interesting or there is simply too much traffic.

comp.graphics
comp.graphics.visualization
comp.graphics.data-explorer
comp.graphics.explorer
comp.graphics.avs
comp.soft-sys.khoros
comp.infosystems.gis

References and Suggested Readings

Banchoff, T.F. (1990). *Beyond the third dimension*. Scientific American Library.

Blattner, M.M., Sumikawa, D.A., & Greenberg, R.M. (1989). Earcons and icons: Their structure and common design principles. *Human-Computer Interaction*, 11-44.

Chernoff, H. (1973). The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association, 68*(342), 361-368.

Clark, J.H. (1982). The geometry engine: A VLSI geometry system for graphics. *Computer Graphics, 16*(3),127-133.

Crawford, S.L., & Fall, T.C. (1990). Projection pursuit techniques for visualizing high-dimensional data sets. In Nielson, G.M., Shriver, B., & Rosenblum, L.J. (Eds.), *Visualization in scientific computing* (pp.94-108).

Doolen, G.D. (1991). *Lattice gas methods: Theory, applications and hardware*. MIT Press.

Duda, R.O., & Hart, P.E. (1973). *Pattern classification and scene analysis*. John Wiley & Sons.

Dunne S., Napel, S., & Rutt, B. (1990). Fast reprojection of volume data. *Proceedings of the First Conference on Visualization in Biomedical Computing* (pp. 11-18).

Ellson, R., & Cox, D.(1988). Visualization of injection molding. *Simulation, 51*(5),184-188.

Foley, J.D., et al. (1990). *Computer graphics: Principles and practice* (2nd ed.). Addison Wesley.

Freeman, W.T., Adelson, E.H., & Heeger, D.J. (1991). Motion Without Movement. *Computer Graphics, 25*(4),27-30.

Friedman, J.H. (1987). Exploratory projection pursuit. *Journal of the American Statistical Association, 82*(397), 249-266.

Fuchs, H., et al. (1989). Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics, 23*(3),79-88.

Grinstein, G., Pickett, R.M., & Williams, M.G. (1989). EXVIS: An exploratory visualization environment. *Proceedings of Graphics Interface* (pp. 254-261).

Haber, R.B., & McNabb, D.A. (1990). Visualization idioms: A conceptual model for scientific visualization systems. In Nielson, G.M., Shriver, B. & Rosenblum, L.J. (Eds.), *Visualization in scientific computing* (pp. 74-93).

Helman, J., & Hesselink, L. (1991). Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications, 11*(3), 36-46.

Hsiung, P.K., Thibadeau, R.H., & Dunn, R.H.P. (1990). Ray-tracing relativity. *Pixel, 1*(1), 10-18.

Hultquist, J.P.M. (1990). Interactive numerical flow visualization using stream surfaces. *Computing Systems in Engineering, 1*(2), 349-353.

Kajiya, J.T. (1986). The rendering equation. *Computer Graphics, 20*(4), 143-149.

Kamada, T. (1989). Visualizing abstract objects and relations: A constraint-based approach. *World Scientific Publishing.*

Kaufman, A. (1991). Volume visualization. *IEEE Computer Society Press Tutorial.*

Kaufman, A., & Bakalash, R. (1988). Memory and processing Architecture for 3D Voxel-based imagery. *IEEE Computer Graphics and Applications, 8*(11), 10-23.

Krueger, W. (1991). Volume Rendering and Data Feature Enhancement. In *Siggraph'91: State of the Art in Volume Visualization Course Notes*, IV-16 to IV-23.

Laur, D., & Hanrahan, P. (1991). Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics, 25*(4), 285-288.

Levoy, M. (1988). Display of surfaces from volume data. *IEEE Computer Graphics and Applications, 8*(5), 29-37.

Lorensen, W.E., & Cline, H.E. (1987). Marching cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics, 21*(4), 163-169.

McCormick, B.H., DeFanti, T.A., & Brown, M.D. (1987). Visualization in scientific computing. *Computer Graphics, 21*(6).

Meinzer, H.P., et al. (1991). The Heidelberg ray tracing model. *IEEE Computer Graphics and Applications, 11*(6), 34-43.

Mendez, R.H. (1990). *Visualization in supercomputing*. Springer-Verlag.

Nielson, G.M., & Hamann, B. (1990). Techniques for the interactive visualization of volumetric data." *Proceedings Visualization '90* (pp. 45-50).

Nielson, G.M., Shriver, B., & Rosenblum, L.J. (1990). *Visualization in scientific computing*. IEEE Computer Society Press.

Onodera, T., & Kawai, S. (1987). *A formal model of visualization in computer graphics systems*. Springer-Verlag.

Pratt, W.K. (1991). *Digital image processing* (2nd ed.). Wiley.

Press, W.H., Flannery, B.P., Teukolsky, S.A., & Vetterling, W.T. (1986). *Numerical recipes: The art of scientific computing*. Cambridge University Press.

Reeves, W.T. (1983). Particle systems—A technique for modeling a class of fuzzy objects. *Computer Graphics, 17*(3), 359-376.

Rogers, D.F. (1985). *Procedural elements for computer graphics*. McGraw-Hill.

Rogers, D.F., & Adams, J.A (1990). *Mathematical elements for computer graphics* (2nd ed.). McGraw-Hill.

Rosenfeld, A., & Kak, A.C. (1982). *Digital picture processing* (2nd ed. vol 1 & 2). AcademicPress.

Simmons, M., & Koskela, R. (1990). *Performance instrumentation and visualization*. ACM Press.

Thalmann, D. (1990). *Scientific visualization and graphics simulation*. Wiley.
Thalmann, N., & Thalmann, D. (1991). *New trends in animation and visualization*. Wiley.
Tufte, E.R. (1983). *The visual display of quantitative information*. Graphics Press.
Upson, C., et al. (1989). The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications, 9*(4), 30-42.
Upson, C., & Keeler, M. (1988). V-buffer:Visible volume rendering. *Computer Graphics, 22*(4), 59-64.
Van Gelder, A., & Wilhelms, J. (1992). Interactive animated visualization of flow fields. *1992 Workshop on Volume Visualization, ACM* (47-54).
Waltz, E.,& Llinas, J. (1990). *Multisensor data fusion*. Artech House.
Watt, A. (1993). *Three-dimensional computer graphics* (2nd ed.). Addison Wesley.
Wejchert, J., & Haumann, D. (1991). Animation aerodynamics. *Computer Graphics, 25*(4), 19-22.
Wenzel, E.M., et al. (1990). A system for three dimensional acoustic visualization in a virtual environment workstation. *First IEEE Conference on Visualization* (pp. 263-272).
Westover, L. (1990). Footprint evaluation for Volume Rendering. *Computer Graphics, 24*(4), 367-376.
Wilhelms, J. (1991). *Decisions in direct volume rendering*. UCSC-CRL-91-12.
Wilhelms, J., & Van Gelder, A. (1990). Octrees for faster iso-surface generation. *Computer Graphics, 24*(5), 57-62.

Captions for Color Prints

Chapter 5
Color Print 1: Raster graphic

Chapter 8
Color Print 2: CAChe Vitamin B_{12} stereo pair
Color Print 3: CAChe Hexacarbonyls
Color Print 4: FieldView clipping

Chapter 9
Color Print 5: Three-dimensional graph using the circle function
Color Print 6: Sample output frame from a climate model

Chapter 11
Color Print 7: Effect of color mapping on interpretation
Color Print 8: Water, OH-H bond breaking

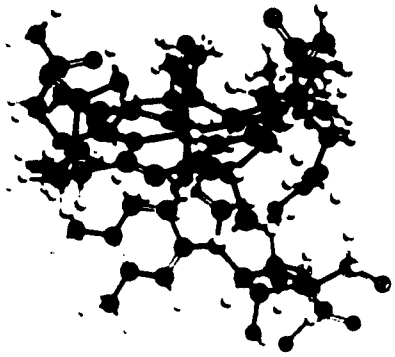
Chapter 12
Color Print 9: Visualization study of a thunderstorm; Visualization Group, NCSA, c. 1989
Color Print 10: Visualization study of the NSFnet; c. Donna J. Cox and Robert Patterson, NCSA

Chapter 13
Color Print 11: The Sudanese Möbius band, lower right, is the initial form generated by the real-time interactive computer animation illiSnail. It is related to the "standard" form of the Möbius band, lower right, as the spherical bead, upper right, is to an untwisted annular band spanning two unlinked (yellow) circles. Retract the circular ribs connecting opposite points on the curve(s). These shapes appear in the solution to Exercise 1.1 at the end of Chapter 13 of this book.

Color Print 12: A once-twisted annulus spans two linked circles, upper left. Stretch the ribs connecting opposite points on the curves to full circles and thus generate a torus, upper right, by a succession of pairwise linked circles. This is a round shadow of the flat, Clifford torus in 4-space. At the lower right is a Möbius band with 3 half-twists, spanning a yellow trefoil knot. Move the knot to a circle and obtain a shadow of Lawson's minimal Kleinbottle in 4-space, which is also a smooth form of Brehm's knotbox. These shapes appear in the solution to Exercises 2 and 3 at the end of Chapter 13 of this book.

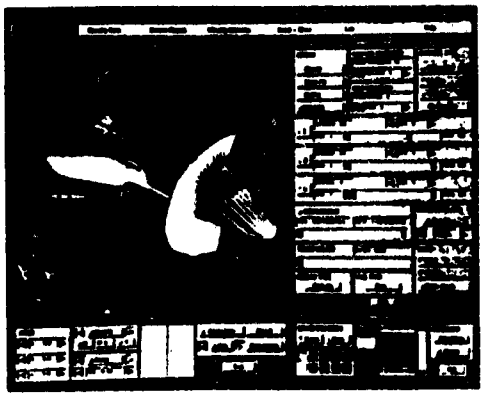
Color Prints 11 and 12 were composed by Chris Hartman in Adobe Photoshop with frames from a Silicon Graphics Iris 4D/300VGX and printed on a Tektronix Phaser II color printer in the Renaissance Experimental Laboratories of the NCSA, University of Illinois. ©George Francis, Chris Hartman, and Glenn Thappell, NCSA, University of Illinois, 1993.

Color Print 1

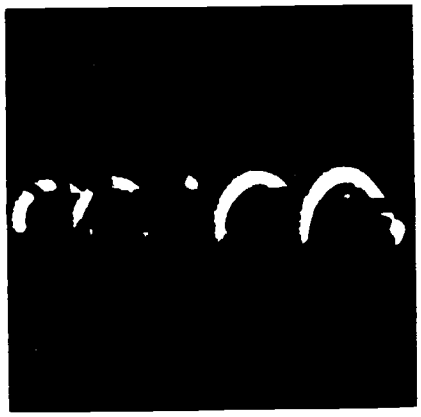


Color Print 2

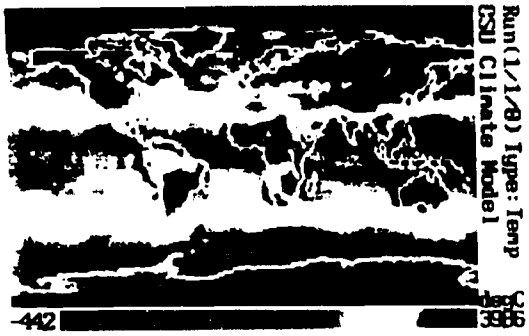
Color Print 3



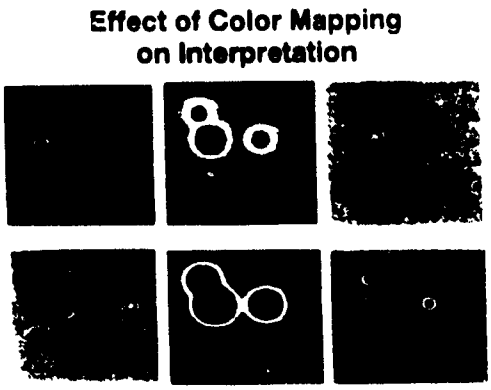
Color Print 4



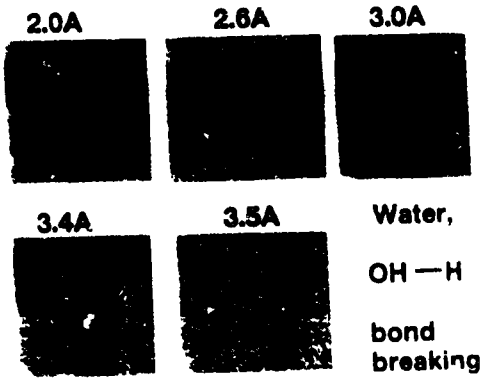
Color Print 5



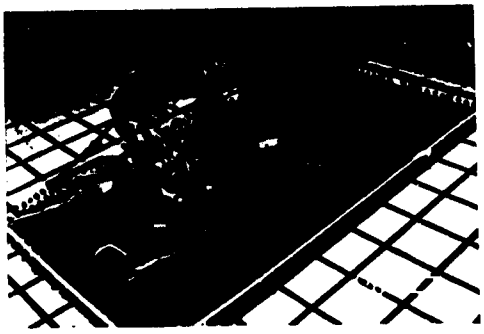
Color Print 6



Color Print 7

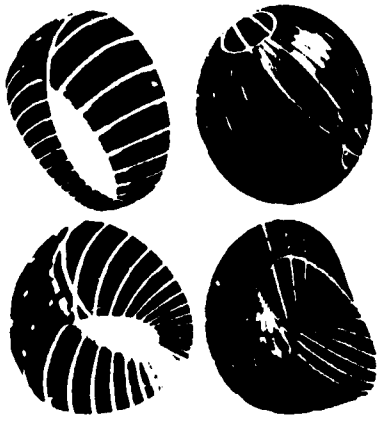


Color Print 8



Color Print 9

Color Print 10



Color Print 11

Color Print 12

AACE

ASSOCIATION FOR THE ADVANCEMENT OF COMPUTING IN EDUCATION

ISBN 1-880094-09-6

BEST COPY AVAILABLE